Programación, depuración del algoritmo SSVD en FORTRAN para el cálculo de valores y vectores propios de una matriz simétrica con alta precisión relativa

Germán Villanueva Baschwitz
Departamento de Matemáticas
Universidad Carlos III de Madrid
28911-Leganés, España

30 de junio de 2009

PROYECTO FIN DE CARRERA
INGENIERÍA INDUSTRIAL SUPERIOR
TUTOR: JUAN MANUEL MOLERA MOLERA

A Juan Manuel Molera
por su dedicación e infinita paciencia

A Laura (cabecita de coco)
por aguantarme, apoyarme
y lo que es más difícil, quererme

Índice

1. Introducción

1.1. Objetivo del proyecto

En el año 2003, Froilán M. Dopico, Juan M. Molera y Julio Moro de la Universidad Carlos III de Madrid, presentan formalmente a la comunidad matemática a través de un artículo de SIAM (Society for Industrial an Applied Mathematics) el algoritmo SSVD (Signed Singular Value Descomposition) [6]. Se trata de un algoritmo basado en un innovador planteamiento que resuelve la descomposición espectral de una matriz simétrica con alta precisión relativa.

El problema que aborda el algoritmo SSVD, la descomposición espectral, o lo que es lo mismo, el cálculo de los valores y vectores propios de una matriz, es un problema que se presenta de manera recurrente campos muy diversos, desde el estudio de fenómenos de resonancia, cálculo de estructuras, pasando por simulaciones de mecánica de fluidos, análisis estadístico de poblaciones, hasta en algoritmos de búsqueda (Google), es por tanto un problema de gran interés práctico.

El problema de valores propios ha sido y continúa siendo objeto de intensivo estudio dentro del Álgebra Lineal Numérica porque, a pesar de existir algoritmos muy potentes para resolverlo, sigue siendo un problema abierto. Los algoritmos tradicionales existentes para la obtención de la descomposición espectral funcionan muy bien para la inmensa mayoría de las matrices (entre ellos, quizás el más importante es el algoritmo QR) [10, 4], pero para determinadas clases de matrices, particularmente "delicadas", estos algoritmos no pueden garantizar la obtención de los valores y vectores propios con una precisión deseable, llegando incluso a proporcionar soluciones al problema totalmente erróneas, valores y vectores propios sin ninguna cifra significativa correcta.

Tanto el algoritmo SSVD, objeto de estudio del presente trabajo, como el algoritmo J-Orthogonal [13, 11], al que nos referiremos a menudo, así como el reciente Jacobi implícito [7, Implicit Standard Jacobi Gives High Relative Accuracy] pertenecen a una nueva generación de algoritmos que pueden garantizar el cálculo de la descomposición espectral en casos muy desfavorables, matrices para las que no se podía resolver la descomposición espectral satisfactoriamente utilizando los algoritmos convencionales.

El algoritmo SSVD tal y como se presentó en 2003, pese a los importantes resultados que con él se obtuvieron (mejorando los convencionales), no era todo lo preciso que debiera en el calculo de los vectores propios para algunas distribuciones muy determinadas de valores propios. Finalmente a lo largo de 2006 los autores del algoritmo SSVD original resuelven esa limitación introduciendo un serie de modificaciones [5].

El presente proyecto puede resumirse en tres etapas o tres grupos de actividades en las que el alumno ha desarrollado su trabajo:

- Implementación y depuración del algoritmo SSVD definitivo en lenguaje FORTRAN.

  La implementación se ha llevado a cabo en el entorno VISUAL STUDIO 2005 de Microsoft. Se ha utilizado el lenguaje FORTRAN por su extendido uso en Álgebra Lineal Numérica para la programación de algoritmos formales debido a su eficiencia y potencia. Es también el lenguaje en el que están escritas las rutinas de la librería LAPACK (Linear Algebra Package). A lo largo del proyecto también se ha hecho uso de MATLAB como herramienta auxiliar por su mayor versatilidad y sencillez de utilización.

- Realización de experimentos numéricos con el algoritmo SSVD.

  Se han realizado extensivos experimentos numéricos con el algoritmo SSVD. Como consecuencia se ha obtenido la descomposición espectral para varios centenares de miles de matrices, matrices generadas aleatoriamente controlando diversos parámetros y matrices especialmente diseñadas para llevar al límite la precisión del algoritmo SSVD. Los resultados han sido analizados cuidadosamente comprobando que reproducen fielmente lo que la teoría predice.

- Presentación y documentación del algoritmo SSVD.

  El trabajo que se resume en este documento:

  - Presentación del algoritmo SSVD a la comunidad no especializada.
  - Repaso de los principios fundamentales del Álgebra Lineal (AL) y del Álgebra Lineal Numérica (ALN) necesarios para comprender el algoritmo.
  - Descripción de los fundamentos en los que el algoritmo SSVD se basa, desde el punto de vista del AL y el ALN.
  - Descripción de los experimentos numéricos realizados y presentación de los resultados.

1.2. Antecedentes

En esta sección introduciremos de una manera más formal el problema que resuelve el algoritmo SSVD, la descomposición espectral. Se tratarán la problemática que pueden surgir a la hora de resolverlo y de la necesidad de desarrollar algoritmos que proporcionen alta precisión relativa como el SSVD.

De ahora en adelante se asumirá que al lector le son familiares los principios básicos del Álgebra Lineal y el Álgebra Lineal Numérica. En la sección §2 el lector dispone de un glosario de los términos y principales conceptos del AL y ALN a los que se acudirá con frecuencia a lo largo del documento. En la medida de lo posible, para facilitar al lector la lectura de esta sección introductoria, trataremos de describir someramente cada concepto que se introduzca, adicionalmente se dará la referencia de los que se encuentren descritos más detalladamente en la sección §2 o en otro lugar del documento.

El cálculo de los valores propios de una matriz cualquiera equivale obtener las raíces del polinomio característico $p_A(z) = det(zI - A)$, se trata de un polinomio de grado igual a la dimensión de la matriz. La expresión $p_A(z) = 0$ se conoce como la ecuación característica y sus soluciones $z_1, \ldots, z_n$, son precisamente los valores propios $\lambda_1, \ldots, \lambda_n$. Évariste Galois demostró en 1830 que una ecuación polinómica de grado 5 o mayor no puede ser resuelta, en general, explícitamente, por lo tanto no existen métodos directos para calcular los valores propios, se ha de acudir a métodos iterativos.

Un método iterativo trata de resolver un problema mediante aproximaciones sucesivas a la solución, empezando, a veces, desde una estimación inicial. Idealmente un algoritmo podría converger a la solución exacta del problema a través de un número infinito de iteraciones, en este caso a un valor propio exacto λ. En la práctica esto no sucede así, los métodos iterativos se implementan en algoritmos que son ejecutados en una máquina o un ordenador que sólo puede realizar un número finito de iteraciones. Por lo tanto, en general, no es posible encontrar la solución exacta λ, se trata de calcular una aproximación a la solución del problema $\widehat{\lambda}$, controlando

el error que se comete en el cálculo.

El algoritmo tratado en este trabajo calcula la descomposición espectral de una matriz para el caso simétrico, comencemos pues por definir lo que en Álgebra Lineal se entiende por la descomposición espectral de una matriz simétrica (ver también sección §2.1).

Definición 1.1 *Una* descomposición espectral *de una matriz real simétrica* $A \in \mathbb{R}^{n\times n}$ *es una factorización:*

$$A = Q\,\Lambda\,Q^T,$$

donde $Q \in \mathbb{R}^{n\times n}$ *es una matriz real ortogonal (*$Q^TQ = \mathbb{I}_n$, $\mathbb{I}_n$ *es la matriz identidad de dimensión* $n \times n$*) y* $\Lambda = diag[\lambda_1, \ldots, \lambda_n]$ *es una matriz diagonal. Consideramos* $\lambda_1 \geq \ldots \geq \lambda_n$*. Las columnas* $q_i,\ i = 1, \ldots, n$*, de* Q *son los vectores propios de* A *que corresponden a los valores propios* $\lambda_i,\ i = 1, \ldots, n$*.*

Un ordenador, al trabajar en aritmética digital con una capacidad de almacenamiento finita, introduce a su vez un error ineludible, que se conoce como la unidad de redondeo ϵ (37), que será determinante para la resolución del problema. El mínimo error relativo que puede cometer un algoritmo al resolver un problema cualquiera es precisamente ϵ. Será crítico controlar la propagación de los errores a lo largo del algoritmo ya que, cada operación aritmética realizada introducirá una imprecisión (ver sección §2.3). Estas y otras vicisitudes que transcienden el problema puramente algebraico, son objeto de estudio de otra disciplina complementaria al Álgebra Lineal que ya hemos mencionado anteriormente, el Álgebra Lineal Numérica.

Los algoritmos convencionales que resuelven la descomposición espectral de una matriz (QR, Divide and Conquer, bisección, $\cdots$) garantizan precisión absoluta. Esto es, para los valores propios,

$$|\widehat{\lambda_i} - \lambda_i| \leq O(\epsilon)||A|| = O(\epsilon)\ \text{máx}\,|\lambda_i|, \tag{1}$$

donde λ_i es el valor propio exacto, $\widehat{\lambda_i}$ el valor propio calculado, ϵ la unidad de redondeo. La norma de la matriz $||A||$ es del orden del máximo valor propio $|\lambda_i|$. Y la expresión, para los vectores propios,

$$\Theta(q_i, \widehat{q_i}) \leq \frac{O(\epsilon)\ ||A||}{min_{i\neq j}|\lambda_i - \lambda_j|} \simeq \frac{O(\epsilon)}{\frac{min_{i\neq j}|\lambda_i - \lambda_j|}{max_j|\lambda_i|}}, \tag{2}$$

donde q_i y $\widehat{q_i}$ son los vectores propios exacto y calculado, respectivamente, y $\Theta(q_i, \widehat{q_i})$ el ángulo que forman entre ellos. La cantidad $min_{i\neq j}|\lambda_i - \lambda_j|$ se conoce como gap absoluto, es la distancia entre el valor propio λ_i y su valor propio más cercano λ_j. Esta cantidad tiene gran importancia a lo hora de calcular los vectores propios de una matriz, ya que estos se podrán obtener de manera más precisa en la medida que los valores propios a los que están asociados se encuentren separados. Dicho de otra manera, podemos esperar que el error cometido en el cálculo de dos vectores propios q_i y q_j cuyos valores propios correspondientes sean muy parecidos $\lambda_j \simeq \lambda_j$, sea grande. Este es un fenómeno muy a tener en cuenta, particularmente relevante en el caso del algoritmo SSVD por razones que se tratarán más adelante.

Debido a que los algoritmos convencionales proporcionan precisión absoluta, para valores propios muy pequeños respecto la norma de la matriz ($\frac{max_j|\lambda_j|}{|\lambda_i|} \sim \frac{1}{\epsilon}$), un algoritmo convencional puede proporcionar aproximaciones de los valores propios pequeños con ninguna cifra significativa correcta, e incluso con el signo incorrecto. Además, si hay dos o más valores propios pequeños, sus vectores propios correspondientes pueden ser computados de manera muy imprecisa, incluso cuando los valores propios pequeños estén considerablemente separados desde un punto de vista relativo (ej: $\lambda_i = 10^{-15}$ y $\lambda_j = 10^{-16}$ si $\lambda_1 = 1$).

Para resolver esta situación son necesarios algoritmos que proporcionen alta precisión relativa (APR), esto es:

$$\frac{|\lambda_i - \widehat{\lambda}_i|}{|\lambda_i|} \leq O(\epsilon) \tag{3}$$

para los valores propios y

$$\Theta(q_i, \widehat{q}_i) \leq \frac{O(\epsilon)}{relgap(|\lambda_i|)} \tag{4}$$

para los vectores propios, donde el gap relativo está definido, para cualquier elemento de un conjunto de números reales, como (ver también definición 3.7 sección §3.2.3):

$$relgap(\lambda_i) = min\{min_{i \neq j}(\frac{|\ \lambda_j - \lambda_i\ |}{|\ \lambda_i\ |}, 1)\}.$$

Es el gap mencionado anteriormente pero desde el punto de vista relativo, es la distancia relativa del valor propio λ_i al valor propio más cercano λ_j. Como se ve en la expresiones de las cotas de errores relativos en el cálculo de los vectores propios, el gap relativo aparece en el denominador, lo que significa que un λ_i con un relgap asociado pequeño, inducirá un error relativo grande en el cálculo de su vector propio correspondiente q_i.

A continuación ilustraremos con un ejemplo la necesidad de disponer de algoritmos que garanticen alta precisión relativa. Pondremos a prueba el algoritmo QR, el algoritmo más importante que resuelve la descomposición espectral de una matriz. Para ello proponemos la matriz simétrica A de dimensión 3 que contiene valores propios que son muy pequeños respecto la norma de la matriz:

$$A = \begin{bmatrix} 10^{40} & 10^{29} & 10^{19} \\ 10^{29} & 10^{20} & 10^{9} \\ 10^{19} & 10^{9} & 10^{1} \end{bmatrix}$$

Matrices de estas características pueden plantearse en aplicaciones habituales, por ejemplo para calcular la distribución de presiones en el ala de un avión o obteniendo los modos de vibración en el diseño de un puente. A efectos prácticos podemos considerar exactos los valores propios calculados con el programa MATLAB trabajando con 50 cifras significativas:

$$\Lambda_{exacto} = diag[\lambda_1,\ \lambda_2,\ \lambda_3] = diag[+1,000000 \cdot 10^{40},\ +9,900000 \cdot 10^{19},\ +9,818181 \cdot 10^{-01}]$$

Estos son los valores propios obtenidos trabajando con MATLAB con 16 cifras significativas (que utiliza el algoritmo QR) y mediante el algoritmo SSVD que proporciona alta precisión relativa:

$$\begin{aligned} \Lambda_{QR} &= diag[+1,000000 \cdot 10^{40},\ -8,100000 \cdot 10^{19},\ -1,208925 \cdot 10^{+24}] \\ \Lambda_{SSVD} &= diag[+1,000000 \cdot 10^{40},\ +9,900000 \cdot 10^{19},\ +9,818181 \cdot 10^{-01}] \end{aligned}$$

Se puede apreciar que el algoritmo QR, que solo puede garantizar alta precisión absoluta, tiene problemas para calcular los valores propios pequeños en relación a la norma de la matriz, valores propios λ_2 y λ_3. Para estos valores propios, el algoritmo QR no puede proporcionar ninguna cifra significativa e incluso obtiene sus signos incorrectamente. Sin embargo el algoritmo SSVD, puede proporcionar alta precisión relativa, no plantea inconveniente alguno a la hora calcular los valores propios de la matriz A. Este pequeño ejemplo deja patente la importancia de contar con algoritmos que trabajen con Alta Precisión Relativa (APR).

En general no es posible obtener siempre alta precisión relativa. Actualmente, sólo es posible obtener la descomposición espectral con alta precisión relativa para unas determinadas clases de matrices simétricas. El problema no simétrico de valores y vectores propios con APR permanece abierto. Podemos resumir el trabajo que se debe realizarse en cuanto a APR se refiere en tres etapas.

- Determinar las clases de matrices y perturbaciones para las que son posible garantizar APR
- Desarrollar una nueva teoría de perturbación adaptada a las clases específicas de matrices.
- Desarrollar nuevos algoritmos que trabajen con los errores multiplicativos adecuados.

Para el desarrollo de algoritmo SSVD se ha pasado por estos tres puntos. No obstante, el proceso se ha visto simplificado debido a que el algoritmo SSVD toma como punto de partida otro problema del ALN que ha sido muy estudiado y es ampliamente conocido, la descomposición en valores singulares o SVD (sección §2.2).

La identificación de las clases de matrices para las que tanto la SVD como la descomposición espectral puedan ser calculadas con alta precisión relativa ha sido un área de investigación muy activa en los últimos 15 años [3]. Pese a que la descomposición SVD y la descomposición espectral son problemas distintos, para el caso simétrico presentan importantes analogías. Esto plantea una situación muy interesante de la que el algoritmo SSVD se aprovecha, ya que se han identificado gran cantidad de clases de matrices, fácilmente caracterizables, para la que las que se puede computar su SVD con APR, así como algoritmos que las calculan. Para la descomposición espectral sin embargo, las matrices no son tan sencillas de reconocer y apenas se han desarrollado algoritmos.

El logro del algoritmo SSVD es aprovechar estos resultados, en lo que APR se refiere, que se conocen de la descomposición SVD para abordar la descomposición espectral. Gracias al algoritmo SSVD se pueden calcular los valores y vectores propios con alta precisión relativa para cualquier matriz simétrica para la que se puede calcular su descomposición SVD de la misma forma [3].

1.3. El algoritmo SSVD

El algoritmo SSVD utiliza las analogías que se presentan en el caso simétrico entre la descomposición espectral y la descomposición SVD (§2.2). Debido a esto el algoritmo SSVD se puede beneficiar de los importante avances que se han conseguido en la descomposición SVD en lo que a alta precisión relativa se refiere. La descomposición SVD es un problema profundamente estudiado, se conoce muy bien para qué matrices y con qué algoritmos esta descomposición puede ser computada con alta precisión relativa.

La importancia del algoritmo SSVD es que consigue computar los vectores y valores propios con alta precisión relativa para todas la matrices simétricas para las que se puede obtener la descomposición SVD de la misma forma (matrices identificadas en [3]). Este algoritmo y el algoritmo de Jacobi implícito [7] a partir de una RRD, son los únicos que alcanzan este grado de generalidad para algoritmos de estas características.

Introduciremos a continuación la descomposición SVD, que como hemos adelantado será de gran importancia para comprender el funcionamiento del algoritmo SSVD.

Definición 1.2 *Dada una matriz $A \in \mathbb{C}^{m\times n}$ la descomposición en valores singulares de A es una factorización que satisface:*

$$A = U\Sigma V^*, \tag{5}$$

donde $U \in \mathbb{C}^{m\times m}$ es unitaria, $V \in \mathbb{C}^{n\times n}$ es unitaria y $\Sigma \in \mathbb{R}^{m\times n}$ es diagonal. Las columnas $u_i, \cdots, u_n$ de U se denominan vectores singulares izquierdos. Las columnas $v_i, \cdots, v_n$ de V se denominan vectores singulares derechos. Los elementos σ_i se denominan valores singulares, estos son no negativos y se representas en orden decreciente $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$, donde $p = min(m, n)$.

El algoritmo SSVD plantea el cálculo de la descomposición espectral de la matriz simétrica en dos dos etapas:

Etapa 1 Obtención de la descomposición en valores singulares de la matriz simétrica A. Esto será realizado por la rutina externa SSYEVJ [8] de la librería LAPACK que implementa el algoritmo one-sided Jacobi.

Etapa 2 Obtención de los valores y vectores propios de A tomando como punto de partida la SVD mediante la rutina `SYSVD`, la rutina que implementará el código fundamental del algoritmo SSVD que será descrita y analizada en detalle en secciones posteriores.

Hagamos aquí una observación importante. Tanto este trabajo como los expuestos en [6] se ocupan principalmente de la etapa 2. La etapa 1 se considerará que es una rutina externa dada. Todos los resultados de errores y códigos presentados se referirán siempre a la segunda etapa, es decir a la rutina `SYSVD`.

El algoritmo SSVD proporcionará alta precisión relativa deseada (3) y (4) en la medida que la descomposición en valores singulares de partida esté computada de la misma forma. Podemos decir que la descomposición SVD se ha obtenido con alta precisión relativa si se satisface, para los valores singulares,

$$|\sigma_i - \widehat{\sigma}_i| = O(\kappa \cdot \epsilon)|\sigma_i| \tag{6}$$

donde σ_i y $\widehat{\sigma}_i$ son los valores singulares exactos y calculados, respectivamente, ϵ la unidad de redondeo y κ una determinada constante moderada, y, para los vectores singulares,

$$\Theta(u_i, \widehat{u}_i) \leq \frac{O(\kappa \cdot \epsilon)}{relgap(\sigma_i)} \tag{7}$$

$$\Theta(v_i, \widehat{v}_i) \leq \frac{O(\kappa \cdot \epsilon)}{relgap(\sigma_i)} \tag{8}$$

donde u_i y v_i son los vectores singulares izquierdos y derechos exactos, respectivamente, $\widehat{u}_i$ y $\widehat{v}_i$ sus correspondientes vectores singulares calculados, y $\Theta(u_i, \widehat{u}_i)$ son los ángulos que forman los vectores exactos y calculados y $relgap$ es la magnitud introducida en la sección anterior (§1.2)

Presentamos a continuación de manera esquemática como el algoritmo SSVD computa la descomposición espectral de una matriz:

Algorithm 1 (SSVD)
INPUT: Matriz simétrica A.
OUTPUT: Valores propios $\Lambda = \mathtt{diag}[\lambda_i]$ y vectores propios $Q = [q_1 \dots q_n]$; $A = Q\Lambda Q^T$.

1. Computar la descomposición SVD $A = U\Sigma V^T$.
2. Computar los valores y vectores propios de la matriz A a partir de los valores y vectores singulares (Algoritmo 3, SYSVD).

En lo que concierne al coste computacional del Algoritmo 1 (SSVD), el número de operaciones necesarias para llevar a cabo el cálculo de la descomposición espectral de una matriz A de dimensión n es de orden $O(n^3)$. La gran mayoría de estas operaciones se concentran en el `paso` 1, cuyo coste computacional es de orden $O(n^3)$, mientras que en el `paso` 2, el coste operacional es de orden $O(n^2)$. Como ocurre habitualmente en los algoritmos que trabajan con alta precisión relativa, el coste computacional es algo mayor que el de los algoritmos convencionales, como el algoritmo QR, divide-and-conquer, $\cdots$

1.4. Organización del trabajo

El resto del documento se estructura de la siguiente manera:

- Sección 2: Donde se revisarán las nociones y conceptos fundamentales del Álgebra Lineal y el Álgebra Lineal Numérica esenciales para entender este trabajo.
- Sección 3: Donde se presentará la primera versión de la rutina `SYSVD`, se describirán los fundamentos en los que se basa desde la perspectiva del Álgebra Lineal y desde el Álgebra Lineal Numérica. Se darán los resultados del análisis de errores y se presentará un ejemplo para ilustrar el funcionamiento del algoritmo SSVD.
- Sección 4: Donde se presentará la versión definitiva de la rutina `SYSVD`, se describirán las mejoras que se han realizado sobre la rutina original. Se darán los resultados del análisis de errores de la rutina `SYSVD` definitiva y se ilustrará con el ejemplo visto en la sección anterior por qué ha sido necesario desarrollar estas mejoras.

- Sección 5: Donde se realizarán experimentos numéricos para poner a prueba el algoritmo SSVD, comprobando que se reproduce lo que la teoría de errores predice. Se realizarán asimismo experimentos con diferentes algoritmos (QR, J-Ortogonal, $\cdots$) para comparar los resultados. Finalmente se compararán las dos versiones de la rutinas `SYSVD` mediante un experimento especialmente diseñado para ilustrar las mejoras.
- Sección 6: Donde se plantearán las conclusiones a las que se ha llegado y se proponen nuevas líneas de investigación dentro del ALN que han sido motivadas por el desarrollo de este trabajo.
- Sección 7: Donde se dan las referencias de los trabajos en los que el alumno se ha apoyado sin las cuales la realización de este proyecto no hubiera sido posible.
- Sección 8: Donde se presentará el código FORTRAN de la rutina `SYSVD`.

2. Nociones y conceptos de Álgebra Lineal y del Álgebra Lineal Numérica

En esta sección revisaremos los principales conceptos fundamentales del Álgebra Lineal y del Álgebra Lineal Numérica esenciales para poder comprender este trabajo. De cualquier modo se asume que el lector dispone de ciertos conocimientos de Álgebra Elemental.

En primer lugar se tratarán el problema del AL que algoritmo SSVD resuelve, la Descomposición Espectral(§2.1).

Posteriormente se tratará otra factorización fundamental de AL, la Descomposición en Valores Singulares o SVD, en la que el algoritmo SSVD se apoya para resolver el caso simétrico de la Descomposición Espectral (§2.2).

Finalmente se introducirán los conceptos de estabilidad y condicionamiento (§2.3) necesarios para caracterizar los problemas matemáticos y los algoritmos que los resuelven. Para ello se realizará un breve recorrido por los fundamentos de la Aritmética en coma flotante.

Para la elaboración de esta sección se han acudido principalmente a las referencias [14], [12] y [4]. Aconsejamos al lector acudir a estas referencias si desea profundizar en los conceptos que esta sección se tratan.

2.1. El problema de valores y vectores propios

Valores y vectores propios

Definición 2.1 *Sea $A \in \mathbb{C}^{n\times n}$ una matriz cuadrada, un vector $x \in \mathbb{C}^n$ distinto de cero es un vector propio de A y $\lambda \in \mathbb{C}$ su correspondiente valor propio si:*

$$Ax = \lambda x \tag{9}$$

La idea es que la acción de la matriz A sobre un subespacio $S_\lambda \in \mathbb{C}^n$ en ocasiones puede reproducir un producto por un escalar. Cuando esto ocurre, ese subespacio S_λ se llama *subespacio propio* y cualquier vector $x \neq 0 \in S_\lambda$ es un vector propio. El conjunto de todos los valores propios de la matriz A se denomina el espectro de A y se denota como $\sigma(A)$.

Polinomio característico

Definición 2.2 *El polinomio característico $p_A(z)$ de una matriz $A \in \mathbb{C}^{n\times n}$, se define como:*

$$p_A(z) = det(zI_n - A), \tag{10}$$

donde I_n es la matriz identidad de dimensión n.

Teorema 2.3 *λ es un valor propio de A si y solo si $p_A(\lambda) = 0$, o lo que es lo mismo, los valores propios son las raíces del polinomio característico.*

Corolario 2.4 *Aunque una matriz sea real, alguno de sus valores propios puede ser complejo.*

Corolario 2.5 *Évariste Galois demostró que una ecuación polinómica general de grado 5 o mayor no puede ser resuelta explícitamente, lo que equivale a decir, que para una matriz de dimensión 5 o mayor no es posible resolver el problema de valores propios de manera explícita, por lo que su resolución se ha de conseguir de manera implícita, mediante métodos iterativos.*

Multiplicidad algebraica

Definición 2.6 *En virtud del teorema fundamental de álgebra podemos escribir p_A en la forma:*

$$p_A(z) = (z - \lambda_1)(z - \lambda_2) \cdots (z - \lambda_n) \tag{11}$$

Según el Teorema (2.3), cada λ_i es un valor propio de la matriz A, y todos lo valores propios han de aparecer en la lista. En general, un valor propio puede aparecer más de una vez. Definimos como multiplicidad algebraica *de un valor propio λ de A como la multiplicidad de λ como raíz del polinomio característico $p_A(z)$. Decimos que un valor propio es simple si su multiplicidad algebraica es igual a 1.*

Subespacios propios

Definición 2.7 *El conjunto de vectores propios asociados a un valor propio λ de una matriz $A \in \mathbb{C}^{n\times n}$, más el vector cero, forman un subespacio que denotaremos por S_λ y que denominaremos subespacio propio. El subespacio propio S_λ se puede representar con la siguiente expresión*

$$S_\lambda = Ker(A - \lambda I_n),$$

donde Ker representa el núcleo.

Definición 2.8 *Llamamos multiplicidad geométrica de $\lambda \in \sigma(A)$ a*

$$\mu(\lambda) = dim(S_\lambda) = dim(Ker(A - \lambda I_n)).$$

La multiplicidad geométrica puede interpretarse como el máximo número de vectores propios linealmente independientes asociados a un valor propio λ.

Teorema 2.9 *La multiplicidad algebraica siempre es mayor o igual que la multiplicidad geométrica.*

Subespacios invariantes

Definición 2.10 *Decimos que V es un* subespacio invariante *bajo la matriz $A \in \mathbb{C}^{n\times n}$ si $\forall x \in V, Ax \in V$.*

Teorema 2.11 *Si V es un subespacio invariante bajo la matriz $A \in \mathbb{C}^{n\times n}$, de dimensión $dimV = p \leq n$, y las primeras p columnas de la matriz invertible $P \in \mathbb{C}^{n\times n}$ son una base de V, entonces*

$$P^{-1}AP = \begin{bmatrix} H & B \\ 0 & C \end{bmatrix}$$

con $\sigma(A) = \sigma(H) \cup \sigma(B)$.

Tanto los subespacios propios, como las sumas directas de subespacios propios son ejemplos de subespacios invariantes.

Transformación de semejanza

Definición 2.12 *Decimos que dos matrices* $A \in \mathbb{C}^{n\times n}$ *y* $B \in \mathbb{C}^{n\times n}$ *son semejantes si existe una matriz* $X \in \mathbb{C}^{n\times n}$ *no singular tal que*

$$B = X^{-1}AX \tag{12}$$

Dos matrices semejantes comparten muchas de sus propiedades.

Teorema 2.13 *Si* A *y* B *son semejantes, entonces tienen el mismo polinomio característico, los mismos valores propios y las mismas multiplicidades geométricas y aritméticas.*

Matrices diagonalizables

Definición 2.14 *Una matriz* A *es diagonalizable, si existe una matriz* $X \in \mathbb{C}^{n\times n}$ *no singular y una matrix* Λ *diagonal tal que*

$$A = X\Lambda X^{-1}. \tag{13}$$

Si una matriz es diagonalizable, a la expresión (13) se la conoce con el nombre de descomposición espectral de A.

Una matriz A es diagonalizable si se puede encontrar una base de $\mathbb{C}^{n\times n}$ respecto a la cual se puede representar como una matriz diagonal. Los vectores que forman la base son las columnas de la matriz X, que son vectores propios de A, y los elementos diagonales de la matriz D son los valores propios de A. La definición (13) se puede reescribir de la siguiente manera:

$$AX = X\Lambda. \tag{14}$$

Si llamamos x_i es la columna i-esima de X y λ_i el elemento i-esimo de la diagonal de Λ, a través del producto de las matrices a ambos lados de la igualdad se tiene que :

$$Ax_i = x_i\lambda_i, \tag{15}$$

donde x_i es un vector propio de A y λ_i, el elemento i-ésimo de la diagonal de Λ es su valor propio correspondiente.

Teorema 2.15 *Una matriz es diagonalizable si y sólo si las multiplicidades geométricas y algebraicas de todos sus valores propios sean iguales.*

Diagonalización unitaria/ortogonal

A veces ocurre que los vectores propios de una matriz A pueden elegirse de tal modo que sean ortonormales. Se dice que $A \in \mathbb{C}^{n\times n}$ es unitariamente diagonalizable si existe una matriz $Q \in \mathbb{C}^{n\times n}$ unitaria, $Q^*Q = QQ^* = I_n$, tal que:

$$A = Q\Lambda Q^* \tag{16}$$

con Λ diagonal y Q^* la traspuesta conjugada de Q.

Definición 2.16 *Diremos que $A \in \mathbb{R}^{n\times n}$ es diagonalizable ortogonalmente si existe una matriz $Q \in \mathbb{R}^{n\times n}$ ortogonal, $Q^TQ = QQ^T = I$, tal que:*

$$A = Q\Lambda Q^T \tag{17}$$

con Λ diagonal.

Teorema espectral

Teorema 2.17 *Una matriz A es unitariamente diagonalizable si y solo si es normal, esto es $A^*A = AA^*$.*

$$A = Q\Lambda Q^* \Leftrightarrow A^*A = AA^* \tag{18}$$

Corolario 2.18 *Los valores propios de una matriz simétrica son reales*

Corolario 2.19 *Los valores propios de una matriz unitaria u ortogonal tienen módulo 1*

De las dos anteriores propiedades se deduce la siguiente,

Corolario 2.20 *Los valores propios de una matriz real simétrica y ortogonal son ± 1*

2.2. La descomposición en valores singulares (SVD)

La descomposición en valores singulares (Definición 1.2), es una factorización cuya computación se plantea en muchos algoritmos más complejos, (como en caso que nos ocupa, el algoritmo SSVD). Igualmente importante es la utilización de la SVD para comprender más profundamente otros problemas del Álgebra Lineal.

Teorema 2.21 *Toda matriz $A \in \mathbb{C}^{m\times n}$ tiene una descomposición en valores singulares. Además los valores singulares σ_i están unívocamente determinados.*

La utilidad de la SVD queda patente cuando se catalogan su relación con otros tópicos fundamentales del Álgebra Lineal.

Propiedades 2.22 *Sea $A \in \mathbb{C}^{m\times n}$, p el mínimo entre m y n y sea $r \leq p$ el número de valores singulares distintos de cero. Entonces*

1. *Si la matriz A es cuadrada y los σ_i son todos distintos, los vectores singulares izquierdos u_i y los vectores singulares derechos v_i están unívocamente determinados salvo por un signo (en el caso complejo por un factor de valor absoluto la unidad).*

2. *El rango de A es r, el número de valores singulares distintos de cero*

3. *Los valores singulares distintos de cero de A son los cuadrados de los valores propios no nulos de AA^* o A^*A (Estas matrices tienen los mismos valores propios, salvo los nulos)*

4. *Si $A = A^*$, los valores singulares de A son los valores absolutos de los valores propios de A.*

Vamos a utilizar las propiedades presentadas anteriormente para llegar a un resultado que posteriormente será necesario para comprender el álgebra en que se basa el algoritmo SSVD.

Teorema 2.23 *Sea $A \in \mathbb{R}^{n\times n}$ una matriz simétrica, $A = A^T$, y sea $A = U\Sigma V^T$ una SVD de A, con $\sigma_1 > \sigma_2 > \ldots > \sigma_p$, los valores singulares distintos y m_i, $i = 1, \ldots, p$ sus respectivas multiplicidades, de modo que $\Sigma = diag[\sigma_1 I_{m_1}, \ldots, \sigma_p I_{m_p}]$ y*

$$U = \left[\, U_1 \,\middle|\, U_2 \,\middle|\, \cdots \,\middle|\, U_p \,\right], \tag{19}$$
$$V = \left[\, V_1 \,\middle|\, V_2 \,\middle|\, \cdots \,\middle|\, V_p \,\right] \tag{20}$$

donde $U_i, V_i \in \mathbb{R}^{n\times m_i}$ son los las columnas que contienen los vectores singulares que corresponden a los valores singulares σ_i. Entonces la matriz V^TU es una matriz diagonal por bloques,

$$V^TU = diag[V_1^TU_1, \ldots, V_p^TU_p]. \tag{21}$$

ortogonal y simétrica.

Demostración: Si aplicamos una transposición a ambos lados de la expresión (5), llegamos a la siguiente expresión:

$$A^T = V\Sigma U^T \tag{22}$$

Multiplicando las expresiones (5) y (22) obtenemos:

$$A^TA = V\Sigma U^TU\Sigma V^T = V\Sigma^2V^T$$

Entonces se tiene que

$$V(A^TA)V^T = VA^2V^T = \Sigma^2$$

De la misma manera se podría haber llegado a la expresión análoga para U

$$UA^2U^T = \Sigma^2$$

Tanto U como V contienen los vectores propios que diagonalizan ortogonalmente A^2

$$\sigma(A^2) = \{\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\} \tag{23}$$
$$A^2v_i = \sigma_i^2 v_i; A^2u_i = \sigma_i^2 u_i \tag{24}$$

El producto escalar de vectores propios de matrices normales que corresponden a distintos valores propios es 0, y por tanto

$$V_i^TU_j = 0, \quad \forall i \neq j$$

Por lo tanto queda demostrado que la matriz V^TU es una matriz diagonal por bloques, estos bloques $V_i^TU_i$ se corresponden con los valores singulares con multiplicidad mayor que uno, de no darse esta circunstancia la matriz V^TU será diagonal, ya que todos los bloques tendrán dimensión igual a la unidad.

Finalmente la matriz V^TU es ortogonal por ser producto de matrices ortogonales. Para demostrar que es simétrica, multiplíquese $A = U\Sigma V^T$ por la izquierda por VV^T,

$$A = V(V^TU\Sigma)V^T \Longrightarrow V^TAV = V^TU\Sigma = \text{diag}[\sigma_1 V_1^T U_1, \ldots, \sigma_p V_p^T U_p] \tag{25}$$

Por ser V^TAV una matriz simétrica, los bloques $V_j^TU_j$ de V^TU también lo son, y por lo tanto ella misma.

■

La SVD y la descomposición espectral

La SVD se puede interpretar como una factorización diagonal Σ de la matriz A cuando el su rango se expresa en la base formada por las columnas de U y su dominio se expresa en la base formada por las columnas de V. Esta interpretación recuerda la descomposición espectral. Veamos las principales diferencias y paralelismos que existen entre factorizaciones:

- La descomposición en valores singulares utiliza dos juegos de vectores (vectores singulares izquierdos u derechos) mientras que la descomposición espectral sólo utiliza uno (los vectores propios).
- La descomposición SVD utiliza bases ortonormales mientras que la descomposición espectral utiliza bases que, en general, no son ortonormales. En el caso particular de las matrices simétricas, al ser ortogonalmente diagonalizables (17), su descomposición espectral si utilizará bases ortonormales.
- No todas las matrices tienen descomposición espectral, pero todas las matrices (incluso las rectangulares) tienen SVD. Pero como hemos visto, todas las matrices simétricas tienen descomposición espectral.

2.3. Condicionamiento y estabilidad

En esta sección abordaremos la discusión de dos tópicos muy relevantes dentro del análisis numérico, el *condicionamiento* y la *estabilidad*:

- El condicionamiento refiere al comportamiento bajo perturbaciones de un determinado problema matemático.
- La estabilidad refiere al comportamiento del algoritmo con el que se resuelve el problema matemático

Para tratar la estabilidad de un algoritmo introduciremos nociones de lo que es la aritmética en coma flotante.

2.3.1. Condicionamiento

Podemos visualizar la resolución de problema como una función $f : X \to Y$ donde X es el espacio vectorial de los datos de entrada e Y es un el espacio vectorial de soluciones del problema. Normalmente la función f no es lineal (incluso en Álgebra Lineal) pero la mayor parte de las veces es una función continua.

Típicamente nos interesara conocer el compartimiento de f en el entorno de un determinado punto $x \in X$ (el comportamiento variará considerablemente en diferentes puntos). Consideramos el problema como la combinación de la función f y datos de entrada x.

Diremos que un problema está bien condicionado si pequeñas perturbaciones de la entrada x producen pequeñas cambios a la salida $f(x)$. Por otro lado diremos que un problema esta mal condicionado si pequeños perturbaciones a la entrada x producen cambios grandes a la salida $f(x)$.

Número de condición absoluto y relativo

Definición 2.24 *Sea δx una pequeña perturbación de x, y $\delta f = f(x+\delta x) - f(x)$ el cambio en la solución producido por la perturbación δx. Se define como número de condición absoluto:*

$$\widehat{\kappa} = \lim_{\xi \to 0} sup\{\frac{\|\delta f\|}{\|\delta x\|} : \|\delta x\| \ \leq \xi\} \tag{26}$$

Si estamos interesados en los cambios relativos, es necesario evaluar de manera relativa. El número de condición relativo κ se define como:

$$\kappa = \lim_{\xi \to 0} sup\{\frac{\|\delta f\|/\|f\|}{\|\delta x\|/\|x\|} : \delta f = f(x+\delta x) - f(x) : \|\delta x\| \ \leq \xi\} \tag{27}$$

Condicionamiento de la multiplicación matriz por vector

Sea $A \in \mathbb{R}^{m \times n}$ y considérese el problema de computar Ax a partir de un vector de partida x. Vamos a determinar en número de condición de este problema $f(x) = Ax$. A partir de la definición de número de condición relativo (27) tenemos que:

$$\frac{\|\delta f\|/\|f\|}{\|\delta x\|/\|x\|} = (\frac{\|A(x+\delta x) - Ax\|}{\|Ax\|} / \frac{\|\delta x\|}{\|x\|}) = \frac{\|A\delta x\|}{\|\delta x\|} / \frac{\|Ax\|}{\|x\|} \leq \|A\| \frac{\| \ x\|}{\|Ax\|}$$

$$\kappa \leq \|A\| \frac{\| \ x\|}{\|Ax\|} \tag{28}$$

Esta es una cota para κ, dependiente tanto de x como de A. Si suponemos que A es una matriz cuadrada y no singular, podemos utilizar el hecho de que $x = A^{-1}Ax \Rightarrow \|x\| \leq \|A^{-1}\|\|A\| \Rightarrow \|\frac{\| \ x\|}{\|Ax\|} \leq \|A^{-1}\|$ para encontrar una cota superior de κ independiente de x.

$$\kappa \leq \|A\|\|A^{-1}\| \tag{29}$$

Esta cota es alcanzable [14] porque existen $x : \|Ax\| = \|A^{-1}\|\|Ax\|$, y $\delta x : \|A\delta x\| = \|A\|\|\delta x\|$, y por tanto

$$\kappa = \|A\|\|A^{-1}\|.$$

Condicionamiento de la solución de un sistema de ecuaciones

Sea $A \in \mathbb{R}^{n\times n}$ no singular y considérese la ecuación $Ax = b$. El problema de computar x, dado un vector b, tiene un número de condición, respecto a las perturbaciones de b, $\kappa = \|A\|\|A^{-1}\|$, puesto que $x = A^{-1}b$.

Número de condición de una matriz

El producto $\|A\|\|A^{-1}\|$ se presenta tan a menudo que tiene su propio nombre, *el número de condición de una matriz* (referido a una norma $\|\cdot\|$) y se denota $\kappa(A)$

$$\kappa(A) = \|A\|\|A^{-1}\| \tag{30}$$

En este caso el término "número de condición" no se relaciona con un problema sino con una matriz. Si $\kappa(A)$ es pequeño se dice que A esta bien condicionada y si $\kappa(A)$ es grande se dice que A está mal condicionada. Si A es singular es frecuente escribir $\kappa(A) = \infty$. Nótese que si la norma $\|.\| = \|.\|_2$ entonces $\|A\| = \sigma_1$ y $\|A^{-1}\| = \sigma_n$ por las propiedades de los valores singulares, y podemos reescribir el número de condición como:

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_n} \tag{31}$$

También podemos escribir el resultado para matrices normales:

$$\kappa_2(A) = \frac{\|\lambda_1\|}{\|\lambda_n\|} \tag{32}$$

2.3.2. Aritmética en coma flotante

Limitaciones de la representación digital

Debido a que un ordenador dispone de un número limitado de bits para representar un número, estos sólo pueden representar una conjunto finito de los numeros reales[1]. Esta limitación presenta dos dificultades:

- Los números representados no pueden ser arbitrariamente grandes o pequeños.
- Entre los números que pueden ser representados habrá saltos, ya que al ser un número finito no se pueden distribuir de forma continua.

Los ordenadores modernos pueden representar números tan grandes y pequeños que la primera de las dificultades difícilmente suele plantearse. Por ejemplo para doble precisión, la mayor y menor cantidad representable, que se conocen como *overflow* y *underflow* respectivamente, son 1.79×10^{308} y 2.23×10^{-308}.

Por otro lado, el problema de los saltos entre los números que pueden representarse es algo que se ha de tener muy presente porque se plantea constantemente. Por ejemplo, en doble precisión, el intervalo $[1, 2]$ se representa por el subconjunto discreto:

$$1, 1 + 2^{-52}, 1 + 2 \times 2^{-52}, 1 + 3 \times 2^{-52}, \cdots, 2 \tag{33}$$

[1]Está sección está basada en la Lección 13 de [12]

El intervalo $[2, 4]$ se representa por los mismos números multiplicados por 2.

$$2, 2 + 2^{-51}, 2 + 2 \times 2^{-51}, 2 + 3 \times 2^{-51}, \cdots, 4 \tag{34}$$

En general, el intervalo $[2^j, 2^{j+1}]$ se representa por subconjunto (33) multiplicado por 2^j. Por esto, en doble precisión aritmética, la máxima distancia relativa entre dos numeros representados nunca será mayor que $2^{-52} \approx 2.22 \times 10^{-16}$. Esta cantidad puede parecer pequeña, y lo es para la mayor parte de las aplicaciones siempre que se utilicen algoritmos estables.

Representación en coma flotante

Este es el sistema universalmente utilizado actualmente. En el sistema de *representación en coma flotante*, la posición punto decimal (o binario) es almacenada separadamente de los dígitos, de esta manera la separación entre los números representados esta escalada con el tamaño de los números. Esto es diferente a lo que ocurría con la representación en coma fija, donde la separación entre números era siempre la misma.

Consideremos un sistema ideal de representación en coma flotante. El sistema consiste en un conjunto discreto de números $\mathbf{F}$ reales determinados por un entero $\beta \geq 2$ que denominados *base* (típicamente 2) y un natural $t \geq 1$ denominado *precisión* (24 y 53 para simple y doble precisión respectivamente). Los elementos de $\mathbf{F}$ son el cero y todos los números de la forma:

$$x = \pm(m/\beta^t)\beta^e, \tag{35}$$

donde m es un natural en el intervalo $1 \leq m \leq \beta^t$ y e es un número entero en el intervalo $e_{min} \leq e \leq e_{max}$. De la misma manera podemos restringir el intervalo de m a $\beta^{t-1} \leq m \leq \beta^t - 1$ siendo entonces la elección de m única. La cantidad $\pm(m/\beta^t)$ se denomina *fracción* o *mantisa* de x, y e el exponente.

Este sistema es ideal porque en principio ignora la restricción del underflow y el overflow, la mínima y máxima cantidad representable respectivamente. Por eso $\mathbf{F}$ tal cual está definido arriba es un conjunto infinito.

Épsilon máquina

La resolución de $\mathbf{F}$ viene dada por un número conocido como *épsilon maquina* que puede definirse como:

$$\epsilon = \frac{1}{2}\beta^{1-t} \tag{36}$$

Este número corresponde con la mitad de la distancia entre 1 y el siguiente número representable en como flotante. Desde un punto de vista relativo es tan grande como puede ser la distancia entre dos números representados en coma flotante. Épsilon maquina tiene la siguiente propiedad.

Para todo $x \in \mathbb{R}$ existe un $x' \in \mathbf{F}$ tal que

$$|x - x'| \leq \epsilon|x|. \tag{37}$$

Tabla 1: Estándar Representación en coma flotante IEEE

$Parámetros - maquina$	$Precisión - sencilla\ (32\ bits)$	$Precisión - doble\ (64\ bits)$
ϵ	$2^{-24} \approx 5,96 \times 10^{-8}$	$2^{-52} \approx 1,11 \times 10^{-16}$
$underflow$	$2^{-126} \approx 1,18 \times 10^{-38}$	$2^{-1022} \approx 2,32 \times 10^{-308}$
$overflow$	$2^{128}(1-\epsilon) \approx 3,40 \times 10^{-38}$	$2^{1024}(1-\epsilon) \approx 1,79 \times 10^{+308}$

Para los valores de β y t más comunes en los ordenadores ϵ se encuentra ente 10^{-16} y 10^{-35}.

Sea $fl : \mathbb{R} \to \mathbf{F}$ una función que da la aproximación mas cercana en coma flotante a un número real cualquiera, la inecuación (37) puede escribirse en términos de fl:

Teorema 2.25 *Para todo $x \in \mathbb{R}$, existe un δ, cumpliendo $|\delta| \leq \epsilon$, tal que $fl(x) = x(1+\delta)$.*

Esto es, la diferencia relativa entre un número real y su representación en coma flotante más cercana siempre es menor que ϵ.

Parámetros máquina estándar IEEE

La gran mayoría de los ordenadores actuales implementan el estándar para coma flotante de la IEEE (The Institute of Electrical and Electronics Engineers). Este estándar especifica de manera precisa la representaciones y operaciones en coma flotante. En la Tabla 1 se presentan los parámetros máquina que hemos visto (underflow, overflow, épsilon maquina) tanto para simple como para doble precisión que se corresponden con el estándar IEEE.

Aritmética en coma flotante

Representar números es el primer paso, lo siguiente que hay que hacer es operar con ellos. En un ordenador todos los cálculos matemáticos se reducen a una cierta cantidad de operaciones elementales, las clásicas con $+, -, \times$ y $\div$. Matemáticamente estos símbolos representan operaciones en $\mathbb{R}$. En un ordenador, de manera análoga son operaciones en $\mathbf{F}$.

Un ordenador está construido en base al siguiente principio. Sean x e y, dos números arbitrarios en coma flotante, $x, y \in \mathbf{F}$ y sea $*$ una de las operaciones elementales $+, -, \times$ y $\div$, entones su correspondecia en coma flotatne $x \circledast y$ debe ser tal que cumpla de manera exacta:

$$x \circledast y = fl(x * y) \tag{38}$$

En virtud de (38) y el Teorema 2.25 se concluye que un ordenador tiene la siguiente simple y potente propiedad:

Axioma 2.26 *AXIOMA FUNDAMENTAL DE LA ARITMÉTICA EN COMA FLOTANTE*

Para todo $x, y \in \mathbf{F}$*, existe un* δ*, con* $|\delta| \leq \epsilon$*, tal que* $x * y = (x \circledast y)(1 + \delta)$

De palabra, toda operación en coma flotante es exacta salvo por un error de tamaño como mucho ϵ.

2.3.3. Estabilidad

Lo ideal sería que un algoritmo pudiera proporcionar la solución exacta a un problema numérico. Los problemas matemáticos suelen ser continuos mientras que los ordenadores trabajan de manera discreta, por lo que esto en general no es posible. La noción de *estabilidad* caracteriza cuales son las limitaciones del algoritmo, diferentes a las limitaciones del problema matemático que son caracterizadas por el condicionamiento.

Algoritmos

Cuando hablábamos de condicionamiento definíamos un problema como una función $f : X \to Y$, siendo X el espacio vectorial de datos e Y el espacio vectorial de soluciones. Un algoritmo se puede ver desde la misma perspectiva, $\widehat{f} : fl(X) \to fl(Y)$ utilizando también los mismos espacios.

Sea un problema f, un sistema de representación en coma flotante que satisface el axioma fundamental 2.26 y un algoritmo $\widehat{f}$ que implemente el problema. Dada un entrada $x \in X$, que puede aproximarse a su representación en coma flotante satisfaciendo el Teorema 2.25, la solución $\widehat{f}(x)$ será un conjunto de puntos que pertenecen al espacio vectorial de soluciones Y, ya que el algoritmo fue diseñado para resolver el problema f.

La situación es delicada ya que el resultado $\widehat{f}(x)$ se vera afectado por los errores de redondeo, convergencias, $\cdots$ no obstante, a pesar de todas estas complicaciones es posible realizar importantes afirmaciones sobre $\widehat{f}(x)$ y por consiguiente de la precisión de los algoritmos del Álgebra Lineal Numérica apoyándose en el axioma fundamental 2.26 y en el Teorema 2.25.

La notación $(\widehat{\cdot})$ es particularmente conveniente, todos los resultados que hayan sido computados y que en general difieren de los resultados exactos se distinguirán con la tilde.

Precisión

Excepto en casos triviales $\widehat{f}(x)$ no puede ser continua. No obstante un buen algoritmo debe aproximarse a su problema asociado f. Para cuantificar esta noción de cercanía, consideraremos tanto el *error absoluto* (39) como el *error relativo*, que cuantificarán lo que llamaremos la precisión absoluta y precisión relativa (40):

$$\|\widehat{f}(x) - f(x)\| \tag{39}$$

$$\frac{\|\widehat{f}(x) - f(x)\|}{f(x)} \tag{40}$$

Lo deseable es que un algoritmo proporcione *alta precisión relativa* (APR) ya que, la precisión absoluta no garantiza un aproximación adecuada cuando se obtienen soluciones que son numéricamente pequeñas desde un punto de vista relativo. No obstante, dado un problema no se puede garantizar la existencia de un algoritmo proporcione APR, por lo que en general no se puede obtener APR.

Hablaremos de un buen algoritmo que si se puede esperar que obtenga la solución con un error relativo pequeño, por pequeño queremos decir de orden ϵ. De manera más formal, el algoritmo $\widehat{f}$ que computa el problema f es *preciso* si para todo $x \in X$ se tiene que:

$$\frac{\|\widehat{f}(x) - f(x)\|}{f(x)} = O(\epsilon) \tag{41}$$

donde $O(\epsilon)$ quiere decir "de orden épsilon máquina".

Estabilidad

Si el problema f está mal condicionado, el error que hemos definido en (41) puede plantearse como un objetivo poco razonable, sin tener en cuenta el error de redondeo ineludible y considerando que los subsiguientes computaciones se resuelven de manera perfecta, está mal condicionamiento puede traer consigo un cambio muy significativo en la solución. Por lo tanto en vez de hablar de precisión en la mayoría de los casos, se recurre a otra noción, la *estabilidad*, que caracteriza de manera más realista lo que está ocurriendo. Diremos que el algoritmo $\widehat{f}$ que computa el problema f es *estable* si para todo $x \in X$ se tiene que:

$$\frac{\|\widehat{f}(x) - f(\widehat{x})\|}{f(\widehat{x})} = O(\epsilon) \tag{42}$$

para un $\widehat{x}$ que cumple que:

$$\frac{\|\widehat{x} - x\|}{x} = O(\epsilon). \tag{43}$$

De palabra, un algoritmo estable proporciona la solución precisa de un problema $\widehat{x}$ próximo al problema exacto x.

Estabilidad regresiva

Algunos algoritmos del álgebra lineal numérica que es mas potente y sencilla que la estabilidad, la *estabilidad regresiva*. Decimos que el algoritmo $\widehat{f}$ que computa el problema f es *estable regresivamente* si para todo $x \in X$ se tiene que:

$$\widehat{f}(x) = f(\widehat{x}) \tag{44}$$

para un $\widehat{x}$ que cumple que:

$$\frac{\|\widehat{x} - x\|}{x} = O(\epsilon). \tag{45}$$

De palabra, un algoritmo estable proporciona la solución exacta de un problema $\widehat{x}$ próximo al problema exacto x.

3. La rutina SYSVD original

En esta sección describiremos el Algoritmo 1 (SSVD) introducido en la sección §1.3, como sabemos se trata de un algoritmo capaz de obtener la descomposición espectral de un matriz simétrica con alta precisión relativa con gran generalidad. Concretamente esta sección se centrará en la primera versión de la segunda etapa del Algoritmo 1 (SSVD) al que los referiremos en lo sucesivo como rutina o algoritmo `SYSVD0`. Adelantamos que sobre ella se realizarán una serie de mejoras para resolver problemas relacionados con la precisión en el cálculo de los vectores propios. La necesidad de realizar estas modificaciones será planteada en la siguiente sección. La rutina `SYSVD` definitiva que implementará esas mejoras será descrito en la sección §4. Consideramos que por claridad en la exposición es necesario introducir previamente esta versión de la rutina, de esta manera al lector le será más sencillo seguir este trabajo.

La rutina `SYSVD0` implementa el planteamiento fundamental, la obtención de la descomposición espectral de una matriz simétrica a partir de su descomposición SVD. Se trata de la segunda etapa del Algoritmo 1 (SSVD) en la que nos centraremos en este trabajo. La primera etapa, la obtención de la descomposición SVD, es como sabemos un problema del Álgebra Lineal Numérica resuelto por lo que será realizada un a rutina externa. Si el lector quiere profundizar en el estudio de los algoritmos que resuelven la descomposición SVD con alta precisión relativa en la bibliografía cuenta con las referencias necesarias [2].

Esta sección seguirá el siguiente esquema:

- Descripción de los fundamentos del Álgebra Lineal en los que se basa la rutina `SYSVD0`.
- Descripción de la rutina `SYSVD0` desde la perspectiva del Álgebra Lineal Numérica. Presentación del pseudocódigo de la rutina `SYSVD0`.
- Presentación de los errores cometidos en el cálculo de los valores y vectores propios.

3.1. Computando la descomposición espectral a partir de la SVD

A continuación describiremos cómo la rutina `SYSVD` obtiene la descomposición espectral a partir de la descomposición SVD aprovechándose de las analogías que ambas descomposiciones presentan para el caso simétrico. El procedimiento por el cual se obtienen los valores y vectores propios es conceptualemnte sencillo y desde el punto de vista del coste operacional, razonablemente económico. En el cálculo de la descomposición espectral a partir de la descomposición SVD se realizan del orden de $O(n^2)$ operaciones frente a las $O(n^3)$ operaciones necesarias para obtener la descomposición SVD.

Aunque se trate el caso general de manera más formal más adelante, comenzaremos describiendo el procedimiento para el caso más sencillo en que todos los valores singulares de la matriz A tienen multiplicidad 1, de manera que todos los subespacios singulares tendrán dimensión 1. Esta situación, en la práctica, se dará en la mayoría de las ocasiones. Los valores singulares σ_i de una matriz simétrica son siempre los módulos de los valores propios λ_i y los vectores propios q_i son los vectores singulares, tanto los derechos como los izquierdos, u_i y v_i, respectivamente:

$$\begin{aligned} \sigma_i &= |\lambda_i| \\ q_i &= u_i \text{ ó } v_i. \end{aligned}$$

Por tanto, la obtención de la descomposición espectral se limita a asignar los signos adecuadamente a los valores singulares para obtener los valores propios y escoger u_i ó v_i como base de los subespacios propios. Esta asignación de signos da nombre al algoritmo SSVD, *Signed Singular Value Decomposition*. Vamos a ver que la información de cuáles son los signos correspondientes de los valores propios está en las matrices U y V, y que esta información es fácilmente accesible.

Si partimos de la descomposición SVD de la matriz simétrica A, $A = U\Sigma V^T$ y multiplicamos a la izquierda por VV^T, que por ser V una matriz ortogonal, $VV^T = I$, es una operación neutra, llegamos a una expresión que guarda un estrecho parecido con la descomposición espectral (9):

$$A = V(V^T U\Sigma)V^T \tag{46}$$

Por ser A una matriz simétrica con valores singulares con multiplicidad 1, $V^T U$ es una matriz diagonal cuyos elementos en la diagonal principal son ± 1, a este resultado se llegó en su forma más general para valores propios con multiplicidad cualquiera en la sección §2.2, Teorema 2.23. Al ser Σ la matriz diagonal que contiene los valores singulares, puede verse que las columnas de la matriz V son los vectores propios de A, ya que forman una base respecto la cual la matriz A se representa como una matriz diagonal, la matriz $V^T U\Sigma$. Los elementos ± 1 contenidos en la diagonal principal de la matriz $V^T U$ son precisamente los signos que estamos buscando.

Éste es el planteamiento que está detrás del algoritmo SSVD, a continuación se describe el procedimiento general, sin restricciones sobre la multiplicidad de los valores singulares. La dificultad que se introduce al poder tener los subespacios singulares dimensión mayor que la unidad es que la matriz $V^T U$ no será diagonal, sino diagonal por bloques (ver Teorema 2.23 en §2.2), los bloques serán submatrices en la diagonal principal del tamaño de la dimensión de los subespacios singulares. De este modo la información de los signos de los valores propios no se obtendrá de manera inmediata y los vectores propios no serán directamente los vectores singulares.

Resolución de la descomposición espectral para el caso general

Sea $A \in \mathbb{R}^{n\times n}$ una matriz simétrica, $A = A^T$, y sea $A = U\Sigma V^T$ una SVD de A, con $\sigma_1 > \sigma_2 > \ldots > \sigma_p$, los valores singulares distintos y m_i, $i = 1, \ldots, p$ sus respectivas multiplicidades, de modo que $\Sigma = diag[\sigma_1 I_{m_1}, \ldots, \sigma_p I_{m_p}]$ y

$$U = \left[\ U_1 \ \middle|\ U_2 \ \middle|\ \cdots \ \middle|\ U_p \ \right], \tag{47}$$
$$V = \left[\ V_1 \ \middle|\ V_2 \ \middle|\ \cdots \ \middle|\ V_p \ \right] \tag{48}$$

donde $U_i, V_i \in \mathbb{R}^{n\times m_i}$ son los las columnas que contienen los vectores singulares que corresponden a los valores singulares σ_i. Ya se vió en el Teorema 2.23 que

$$V^T AV = V^T U\Sigma = \text{diag}[\sigma_1 V_1^T U_1, \ldots, \sigma_p V_p^T U_p] \tag{49}$$

Las submatrices $V_i^T U_i$ contienen la información de los signos de los valores propios $\pm\sigma_i$. Para obtener a partir de la expresión (49) la descomposición espectral de A, será necesario diagonalizar las submatrices ortogonales y simétricas $V_i^T U_i$:

$$V_i^T U_i = W_i J_i W_i^T \tag{50}$$

La diagonalización de estas submatrices es sencilla, al ser ortogonales y simétricas, sus valores propios J_i serán ± 1 (ver Sección §2.2), precisamente los signos de los valores propios de A que estamos buscando, y sus vectores propios W_i serán necesarios para calcular los vectores propios de A. Es adecuado ordenar los valores propios de J_i de la forma:

$$J_i = \begin{bmatrix} +1 & & & & & & 0 \\ & \ddots & & & & & \\ & & +1 & & & & \\ & & & -1 & & & \\ & & & & \ddots & & \\ & 0 & & & & & -1 \end{bmatrix}$$

de modo que podamos reescribir (50) así:

$$V_i^T U_i = [W_i^+ W_i^-] J_i [W_i^{+T} W_i^{-T}] \tag{51}$$

con $W_i^+ \in \mathbb{R}^{m_i \times m_i^+}$. Sustituyendo (50) en (49) tenemos que:

$$\begin{aligned} V^T A V &= \text{diag}[\sigma_1 W_1 J_1 W_1^T, \ldots, \sigma_p V_p^T U_p] = \\ &= \begin{bmatrix} W_1 & & 0 \\ & \ddots & \\ 0 & & W_p \end{bmatrix} \begin{bmatrix} \sigma_1 J_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_p J_p \end{bmatrix} \begin{bmatrix} W_1^T & & 0 \\ & \ddots & \\ 0 & & W_p^T \end{bmatrix} \end{aligned} \tag{52}$$

Si la expresión (52) la multiplicamos a la derecha por W y por W^T por la izquierda, donde

$$W = \begin{bmatrix} W_1 & & 0 \\ & \ddots & \\ 0 & & W_p \end{bmatrix}$$

y simplificamos adecuadamente llegamos a:

$$W^T V^T A V W = Q^T A Q = \text{diag}[\sigma_1 J_1, \ldots, \sigma_p J_p]. \tag{53}$$

La matriz Q es la matriz de los vectores propios $Q = VW = [V_1 W_1 \ldots V_p W_p] = [Q_1 \ldots Q_p]$, donde $Q_i = V_i W_i = V_i [W_i{}^+ W_i^-] = [Q_i^+ Q_i^-]$ y las columnas de la submatriz $Q_i^+ = V_i W_i^+ \in \mathbb{R}^{n \times m_i^+}$ (resp. $Q_i^- = V_i W_i^- \in \mathbb{R}^{n \times m_i^-}$) son la base de subespacio propio correspondiente al valor propio σ_i (resp. $-\sigma_i$) de $A : S_{\sigma_i} = Ker(A - \sigma_i \mathbb{I}_n)$ (resp $S_{-\sigma_i}$). En otras palabras, $A = Q \Lambda Q^T$ con $\Lambda = \text{diag}[\pm \sigma_i]$ es la descomposición espectral[2] de A.

Los vectores propios de A se obtendrán como

$$Q_i = V_i W_i \tag{54}$$

por lo que será es necesario diagonalizar las submatrices $V_i^T U_i$. No obstante si sólo queremos obtener los valores propios podemos utilizar las propiedades de los invariantes de una matriz

[2]Dada la simetría de A, se podría haber llegado a un resultado similar usando los vectores singulares derechos en vez de los izquierdos U_i.

para obtenerlos.

Al ser los valores singulares los módulos de los valores propios tan solo es necesario saber cuales son los signos, esta información está en la matriz diagonal J_i, resultante de diagonalizar los bloques $V_i^T U_i$. Para un valor singular con multiplicidad mayor que uno, la información de cuantos valores propios correspondientes son positivos y negativos está en la diagonal principal de J_i. Basta con conocer la traza, la suma de los elementos de su diagonal principal, para asignar los signos correctamente. Como la traza es un invariante bajo semejanza, podemos obtener la información necesaria de la traza de $V_i^T U_i$ sin necesidad de diagonalizar a través de la fórmula:

$$traza(V_i^T U_i) = traza(J_i) = m_i^+ - m_i^-$$

Si tenemos en mente que $m_i^+ + m_i^- = m_i$ llegamos a la siguiente expresión con la que se obtiene directamente el número de valores propios positivos y negativos dentro de cada valor singular de multiplicidad mayor a 1:

$$m_i^{\pm} = \frac{m_i \pm \mathrm{trace}(V_i^T U_i)}{2}, \tag{55}$$

Todo este planteamiento está implementado en la rutina `SYSVD` original, que corresponde al `paso` 2 del Algoritmo 1 (SSVD). El Algoritmo 2 `(SYSVD0)` se puede describir esquemáticamente en tres pasos que serán descritos detalladamente más adelante:

Algorithm 2 (SYSVD0)

INPUT: Descomposición en valores singulares de una matriz simétrica $A = U\Sigma V^T$
Parámetros: tol < 1, tolgap < 1
OUTPUT: Valores propios $\Lambda = diag[\lambda_i]$ y vectores propios $Q = [q_1 \dots q_n]$; $A = Q\Lambda Q^T$

1. Decidir a través de la tolerancia tol los grupos numéricamente iguales de valores singulares $\{\Sigma_i, U_i, V_i\}_{i=1}^k$ mediante Algoritmo 2.1

2. Calcular los valores propios utilizando el Algoritmo 2.2

3. Computar los vectores propios utilizando el Algoritmo 2.3

3.2. La rutina `SYSVD0`

El punto de partida del algoritmo SSVD es la descomposición en valores singulares de la matriz simétrica A, será necesario conocer en qué condiciones ésta ha sido calculada para analizar con rigor los resultados que proporciona el algoritmo. Comenzaremos suponiendo que la descomposición SVD de la que partimos ha sido computada con pequeños errores multiplicativos, esto es:

Definición 3.1 *Sea* $G = \widehat{U}\widehat{\Sigma}\widehat{V}^T \in \mathbb{R}^{m\times n}$ *la computación de la descomposición SVD de* G *ejecutada con precisión épsilon maquina* ϵ*. Decimos que esta descomposición SVD está calculada con pequeños errores multiplicativos de orden* $\kappa\, O(\epsilon)$ *si existen las matrices* $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, $E \in \mathbb{R}^{m\times m}$, $F \in \mathbb{R}^{n\times n}$ *tales que* U *y* V *tienen columnas ortonormales, y*

$$\begin{aligned} \|U - \widehat{U}\| = O(\epsilon), \quad \|V - \widehat{V}\| = O(\epsilon), \\ \|E\| = \kappa\, O(\epsilon), \quad \|F\| = \kappa\, O(\epsilon), \end{aligned} \tag{56}$$

donde κ *es una constante que depende del condicionamiento de las matrices intermedias, y*

$$(I + E)G(I + F) = U\widehat{\Sigma}V^T \tag{57}$$

Entonces a través los resultados de la teoría de la perturbación que presentamos a continuación [9], podemos garantizar que los valores singulares calculados $\widehat{\Sigma}$ son muy parecidos a los Σ exactos. Además podemos obtener una cota superior del error cometido.

Sea una matriz $G \in \mathbb{R}^{m\times n}$ con una descomposición SVD: $G = U\Sigma V^T$, con los valores singulares $\sigma_1 \geq \sigma_2 \geq \ldots$ Consideramos una perturbación multiplicativa $\widetilde{G} = (I+E)G(I+F)$ de G con SVD: $\widetilde{G} = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T$ y valores singulares $\widetilde{\sigma}_i$, también en orden decreciente. El siguiente teorema de [9, Theorem 3.1] nos da la relación entre los valores singulares de G y $\widetilde{G}$.

Teorema 3.2 *Sea* $G \in \mathbb{R}^{m\times n}$, $\widetilde{G} = (I + E)G(I + F)$, *definiendo* η *y* η' *del siguiente modo:*

$$\eta = \text{máx}\{\|E\|, \|F\|\}, \qquad \eta' = 2\eta + \eta^2. \tag{58}$$

tenemos que

$$\frac{|\,\sigma_i - \widehat{\sigma}_i|}{\sigma_i} \leq \eta'$$

Por tanto, si tenemos una SVD calculada con pequeños errores multiplicativos como en (57) el error relativo en los valores singulares es, según el Teorema 3.2

$$\frac{|\,\sigma_i - \widehat{\sigma}_i|}{\sigma_i} \leq \kappa O(\epsilon) \tag{59}$$

Ahora procederemos a analizar en detalle el algoritmo SYVD0 para el cálculo de la descomposición espectral de una matriz simétrica paso por paso. Se describirá su funcionamiento, y se presentarán tanto el pseudocódigo del algoritmo como los resultados en materia de errores que con él se obtienen.

3.2.1. Paso 1: Selección de agrupaciones de valores singulares

Para obtener de manera satisfactoria la descomposición espectral de una matriz con el Algoritmo 1 (SSVD), será necesario identificar qué valores singulares tienen multiplicidad mayor que uno y cuáles no para resolver cada caso de manera correcta. Al trabajar en aritmética finita es muy improbable que se encuentren dos números exactamente iguales. No obstante, podremos considerar que dos valores propios son numéricamente iguales si son suficientemente parecidos. Será especialmente delicado el criterio para establecer esta frontera entre valores singulares distintos. En esta sección iremos definiendo de manera progresiva cada uno de los conceptos necesarios para comprender esta parte del algoritmo.

En principio, de acuerdo con (59) podemos considerar dos valores singulares numéricamente indistinguibles siempre que:

$$\frac{|\sigma_j - \sigma_{j+1}|}{\sigma_j} \leq C\,\kappa\,\epsilon, \tag{60}$$

para unas determinadas constantes C, κ. En este caso diremos que σ_j y σ_{j+1} pertenecen a la misma **agrupación**. A continuación definiremos de manera más formal el concepto de agrupación tanto para valores singulares como para valores propios al que acudiremos de manera recurrente.

Definición 3.3 *Sea* $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, *siendo* $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$, *una distribución ordenada de numeros reales no negativos. El conjunto* $\{\Sigma_I\}_{I=1}^q$, *se denomina* conjunto de agrupaciones de Σ, *si es una partición de* Σ

$$\Sigma = \bigcup_{I=1}^{q} \Sigma_I, \tag{61}$$

de modo que

1. *Los elementos de cada agrupación son consecutivos,* $\Sigma_I = \{\sigma_{I+1}, \sigma_{I+2}, \ldots, \sigma_{I+d_I}\}$
2. *Los subconjunto* Σ_I *están dispuestos en el siguiente orden: si* $\sigma_a \in \Sigma_I$ *y* $\sigma_b \in \Sigma_J$ *entonces* $\sigma_a > \sigma_b$ *siempre que* $I < J$.

Definición 3.4 *Sea* $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ *un conjunto de números reales,* $\Sigma = \{|\lambda_1|, \ldots, |\lambda_n|\}$ *con* $|\lambda_1| \geq \ldots \geq |\lambda_n| \geq 0$, *y siendo* $\{\Sigma_I\}_{I=1}^q$una agrupación de Σ. *El conjunto* $\{\Lambda_I\}_{I=1}^q$, *se denomina* conjunto de agrupaciones de Λ inducida por $\{\Sigma_I\}_{I=1}^q$ *si*

$$\Lambda_I = \{\lambda_i \in \Lambda : |\lambda_i| \in \Sigma_I\} \tag{62}$$

Obsérvese que $\{\Lambda_I\}_{I=1}^q$ *es una partición de* Λ *y que* $\Lambda_I = \Lambda_I^+ \cup \Lambda_I^-$ *y* $\Sigma_I = \Lambda_I^+ \cup |\Lambda_I^-|$ *con*

$$\Lambda_I^+ = \{\lambda_i \in \Lambda : |\lambda_i| \in \Sigma_I, \lambda_i > 0\} \tag{63}$$
$$\Lambda_I^- = \{\lambda_i \in \Lambda : |\lambda_i| \in \Sigma_I, \lambda_i < 0\}\,. \tag{64}$$

Dado un conjunto Σ, las agrupaciones se pueden definir de varias maneras. En nuestro caso una manera natural es hacerlo a través de la inecuación (60). Nosotros recurriremos más a menudo a la siguiente definición.

Definición 3.5 *Sea* $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, *con* $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$, *una distribución ordenada de números reales no negativos y sea* τ *un número real tal que* $0 \leq \tau < 1$. *El subconjunto* $\Sigma_I = \{\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_{i+d_1}\}$ *de* Σ *es denominado agrupación asociada a una tolerancia* τ *si*

1. $(\sigma_j - \sigma_{j+1}) \leq \tau\,\sigma_j$ *for* $j = i+1, \ldots, i+d_1-1$,
2. $(\sigma_i - \sigma_{i+1}) > \tau\,\sigma_i$ *y* $(\sigma_{i+d_1} - \sigma_{i+d_1+1}) > \tau\,\sigma_{i+d_1}$, *siempre que todos los índices pertenezca a* $\{1, 2, \ldots, n\}$; *de otro modo la inecuación correspondiente no aparece en la definición.*

Dados unos valores singulares calculados $\widehat{\Sigma} = \{\widehat{\sigma}_1 \geq \widehat{\sigma}_2 \ldots \widehat{\sigma}_n\}$ veamos como vamos a seleccionar un conjunto de agrupaciones. Consideramos la función

$$\{\widehat{\Sigma}_i\}_{i=1}^k = \texttt{selectcluster}(\widehat{\Sigma}, \texttt{tol}) \tag{65}$$

que, dados $\widehat{\Sigma}$ y `tol` como entrada, obtiene a la salida, una distribución de agrupaciones $\{\widehat{\Sigma}_i\}_{i=1}^q$ que satisfacen la definición 3.5.

Presentamos a continuación el pseudocódigo correspondiente a esta parte del algoritmo:

Algorithm 2.1 (Primera selección de agrupaciones)
INPUT:

- Valores singulares Σ
- tol: tolerancia para considerar dos Σ_i distintos

OUTPUT:

- Agrupaciones $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$.

```
1.  %Inicializar contadores
2.  r = 0  %n^o de agrupaciones
3.  for i = 1 : n  %n es la dimensión de la matriz A
4.  %Condición para terminar considerar la agrupación Σ_r diferente a Σ_{r-1}
5.    if (σ(i)−σ(i+1))/σ(i) > tol then
6.    %Almacenar donde comienza la agrupación Σ_r (terminará donde comience la siguiente)
7.      agrupacion(r)=i
8.      r = r + 1  %Actualizar contador
9.    endif
10. end for
```

El conjunto de agrupaciones $\{\widehat{\Sigma}_i\}_{i=1}^q$ que es una partición de $\widehat{\Sigma}$, genera una partición equivalente en los vectores singulares calculados $\widehat{U} = [\widehat{U}_1 \ldots \widehat{U}_k]$ y $\widehat{V} = [\widehat{V}_1 \ldots \widehat{V}_k]$. Para cada agrupación $\widehat{\Sigma}_i$ tomamos las matrices $\widehat{U}_i, \widehat{V}_i \in \mathbb{R}^{n \times m_i}$ los vectores izquierdos y derechos correspondientes a los valores singulares $\widehat{\Sigma}_i$.

3.2.2. Paso 2: Obtención de los valores propios

Al tratarse de una matriz simétrica, sabemos que los valores singulares serán los módulos de los valores propios. Por lo tanto, en lo que se refiere al cálculo de los valores propios el Algoritmo 2 (`SYSVD0`) se limitará a asignar los signos adecuadamente a los valores singulares. Por esta razón los valores propios calculados por el Algoritmo 1 (SSVD) tienen exactamente los mismos errores que los valores singulares procedentes de la descomposición SVD que se ha tomado como punto de partida. Dicho de otra manera, en la obtención de los valores propios a partir de los valores singulares no se produce ningún error, los valores propios "heredan"la alta precisión con

la que los valores singulares de partida fueron obtenidos.

La información de los signos de los valores singulares $\widehat{\Sigma}$ esta contenida en las matrices $\widehat{U}$ y $\widehat{V}$. Para obtener el signo de un valor singular con multiplicidad igual a 1, o hablando en términos de aritmética finita, para obtener el signo de una agrupación de valores singulares $\widehat{\Sigma}_i$ de tamaño uno, tan sólo es necesario realizar el producto escalar de los vectores singular izquierdos y derechos $\widehat{u}_i$ y $\widehat{v}_i$ correspondientes. Si la agrupación $\widehat{\Sigma}_i$ tiene un tamaño mayor de uno, se deberá acudir a la expresión (55) para conocer el número de valores propios positivos y negativos, m_i^+ y m_i^-, en la agrupación Σ_i.

Presentamos a continuación (en la siguiente página) el código de forma esquemática correspondiente a esta parte del algoritmo:

Algorithm 2.2 (Valores propios)

INPUT:

- Descomposición SVD de $A = U\Sigma V^T$
- Agrupaciones $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$

OTPUT:

- Valores propios Λ.

1. for *para cada agrupación*, $i = 1 : k$
2. *computar los elementos de la diagonal* $\Delta_i = V_i^T U_i \in \mathbb{R}^{m_i \times m_i}$
3. if $m_i = 1$ then
4. $\lambda_{i_0} = sign(\Delta_i)\sigma_{i_0}$,
5. else
6. for $j = i_0 : i_0 + m_i - 1$
7. $\lambda_j = sign[(\Delta_i)_{jj}]\sigma_j$
8. endfor
9. $t_i = \text{trace}(\Delta_i)$, $m_i^- = \dfrac{m_i - t_i}{2}$
10. if $\#\{(\Delta_i)_{jj} < 0\} \neq m_i^-$ then
11. for $j = i_0 : i_0 + m_i^- - 1$
12. $\lambda_j = -\sigma_j$
13. endfor
14. for $j = i_0 + m_i^- : i_0 + m_i - 1$
15. $\lambda_j = \sigma_j$
16. endfor
17. endif
18. endif
19. endfor

En [6] está demostrado que este procedimiento proporciona valores propios con alta precisión relativa (APR) si la SVD esta calculada con pequeños errores multiplicativos. Los errores cometidos por este algoritmo están expuestos en el siguiente teorema [6, Theorem 4.3]

Teorema 3.6 *Sea A una matriz simétrica real de dimensión $n \times n$ para la que es posible computar la descomposición SVD con pequeños errores multiplicativos de orden $O(\kappa\,\epsilon)$ satisfaciendo (56), (57), y sean $\lambda_1 \geq \ldots \geq \lambda_n$ los valores propios exactos de A y $\widehat{\lambda}_1 \geq \ldots \geq \widehat{\lambda}_n$ las aproximaciones a estos computadas en el* `paso` *1 (con la selección de la distribución de agrupaciones de acuerdo a (65) con $tol = O(\kappa\epsilon)$) del Algoritmo 2* `(SYSVD0)`*. Entonces*

$$|\lambda_j - \widehat{\lambda}_j| = |\lambda_j|\ \kappa\, O(\epsilon) \qquad j = 1, \ldots, n \tag{66}$$

La cota de error (66) se mantiene incluso para los valores singulares nulos, ya que el número exacto de valores singulares nulos de A es conocido cuando se dispone de una descomposición SVD que satisfaga (56)y (57).

3.2.3. Paso 3: Obtención de los vectores propios

La precisión en el cálculo de los vectores propios depende de cómo elijamos las agrupaciones de valores singulares, en esta sección trataremos esta circunstancia en lo que al Algoritmo 2 `(SYSVD0)` respecta. Esta situación motiva las modificaciones que se realizarán sobre el Algoritmo 2 `(SYSVD0)`, éstas serán tratadas en la sección §4 donde se presentará el algoritmo `SYSVD` definitivo.

Sea $\widehat{\Lambda} = \{\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n\}$ un conjunto valores propios calculados mediante el Algoritmo 2 `(SYSVD0)`, tendremos sus correspondientes valores singulares $\widehat{\Sigma} = \{|\widehat{\lambda}_1|, \ldots, |\widehat{\lambda}_n|\}$ que estarán distribuidos en un conjunto de agrupaciones $\{\widehat{\Sigma}_i\}_{i=1}^k$ de $\widehat{\Sigma}$. Esta distribución de agrupaciones inducirá su correspondiente distribución de valores propios $\{\Lambda_i\}_{i=1}^k$. Dicho de otra manera, después de la asignación de signos dentro de cada agrupación de valores singulares, aparecerán dos agrupaciones de valores propios dentro de cada agrupación de valores singulares correspondientes a los valores propios positivos y negativos (solamente una en caso de que todos los valores propios tengan el mismo signo). De acuerdo con la definición 3.4 utilizaremos la notación $\widehat{\Lambda}_i = \widehat{\Lambda}_i^+ \cup \widehat{\Lambda}_i^-$, y $\widehat{\Sigma}_i = \widehat{\Lambda}_i^+, \cup |\widehat{\Lambda}_i^-|$, $i = 1, \ldots, k$. Del mismo modo denotaremos por $S(\Lambda_i^+)$ (resp $S(\Lambda_i^-)$ los subespacios invariantes que son suma directa de subespacios propios correspondientes a los valores propios positivos (resp negativos) dentro de Λ_i^+ (resp Λ_i^-) i.e.

$$S(\Lambda_i^+) = \bigoplus_{\lambda_j \in \Lambda_i^+} Ker(A - \lambda_j \mathbb{I}_n). \tag{67}$$

$$S(\Lambda_i^-) = \bigoplus_{\lambda_j \in \Lambda_i^-} Ker(A - \lambda_j \mathbb{I}_n). \tag{68}$$

entonces los vectores propios computados en el `paso` 4 del Algoritmo 2, $\widehat{Q}_i^+$, (resp $\widehat{Q}_i^-$) son respectivamente aproximaciones a las matrices Q_i^+ (resp Q_i^-), cuyas columnas son bases ortonormales de $S(\Lambda_i^+)$ (resp $S(\Lambda_i^-)$).

Siguiendo las ideas de las sección §3.1, presentamos el pseudocódigo correspondiente a esta parte del algoritmo:

Algorithm 2.3 (VECTORES PROPIOS)
INPUT:

- SVD de $A = U\Sigma V^T$
- Agrupaciones $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$
- Valores propios Λ.

OUTPUT:

- Vectores propios $Q = [q_1 \ldots q_n]$

Notación: Nos referiremos como $Q_i^{\pm}$ al vector propio de la matriz A que se corresponda el valores singular positivo (o negativo respectivamente)

1. for $i = 1 : k$ %*cada agrupación*,
2. if $m_i = 1$ then
3. $q_{i_0} = v_{i_0}$
4. else
5. $\%m_i^- \equiv$ *número de valores singulares negativos en* Σ_i
6. if $m_i^- = 0$ then
7. $Q_i^+ = V_i$
8. elseif $m_i^- = m_i$ then
9. $Q_i^- = V_i$
10. else
11. $\Delta_i = V_i^T U_i$ %*multiplicar*
12. $\Delta_i = [W_i^+ \, W_i^-] J_i [W_i^+ \, W_i^-]^T$ %*diagonalizar*
13. $Q_i^+ = V_i W_i^+, \quad Q_i^- = V_i W_i^-$
14. endif
15. endif
16. endfor

Ya se ha comentado que la precisión con la que los vectores propios son computados depende, entre otras cosas, de las distancias relativas entre agrupaciones de valores singulares, los denominados relgaps. Antes de continuar con el análisis de errores de la computación de los vectores propios detengámonos para definir adecuadamente qué son los denominados relgaps, y cómo se aplican a las distribuciones de valores singulares y valores propios, pues estas consideraciones desempeñarán un papel muy importante en el algoritmo.

Definición 3.7 *Sea una distribución de numeros reales $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ ordenada decrecientemente, $\lambda_1 \geq \cdots \geq \lambda_n$, la distribución de sus módulos, $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, también en orden decreciente, $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$, y sean $\bar{\Lambda} \subseteq \Lambda$ y $\bar{\Sigma} \subseteq \Sigma$ dos subconjuntos de las dos distribuciones, definimos los gaps relativos para los dos subconjuntos de la siguiente manera*

$$relgap(\bar{\Lambda}) = \text{mín}\{\min_{\substack{\lambda_j \in \bar{\Lambda} \\ \lambda_i \notin \bar{\Lambda}}} \frac{|\lambda_i - \lambda_j|}{|\lambda_j|}, 1\}. \tag{69}$$

$$relgap(\bar{\Sigma}) = \text{mín}\{\min_{\substack{\sigma_j \in \bar{\Sigma} \\ \sigma_i \notin \bar{\Sigma}}} \frac{|\sigma_i - \sigma_j|}{\sigma_j}, 1\}. \tag{70}$$

Con estas definiciones podemos describir el error cometido en el cálculo de los vectores propios, el `paso` 2 del Algoritmo 2 `(SYSVD0)` [6, Theorem 4.7]:

Teorema 3.8 *Sea $A \in \mathbb{R}^{n \times n}$ una matriz simétrica de rango r para la que es posible computar la descomposición SVD con pequeños errores multiplicativos de orden $O(\kappa\,\epsilon)$ satisfaciendo (56), (57). Sea $\widehat{\Sigma}_i$ una agrupación de los valores singulares computada en el* `paso` *1, Σ_i la correspondiente agrupación de valores singulares exactos, $\widehat{Q}_i^+, \widehat{Q}_i^-$ los vectores propios computados en el* `paso` *3 del Algoritmo 2* `(SYSVD0)`*, que corresponden respectivamente con los valores propios positivos y negativos $\Lambda_i = \Lambda_i^+ \cup \Lambda_i^-$ computados en el* `paso` *2. Entonces existen matrices Q_i^+ y Q_i^-, cuyas columnas forman una base ortonormal de los subespacios invariantes de A correspondientes a sus valores propios positivos y negativos, donde los valores absolutos de estos valores propios son los valores singulares Σ_i, tales que*

$$\text{máx}\{\| \widehat{Q}_i^+ - Q_i^+ \|_F, \| \widehat{Q}_i^- - Q_i^- \|_F\} \leq \frac{O(\kappa\,\epsilon)}{relgap(\widehat{\Sigma}_i)}. \tag{71}$$

Asimismo, sea $\widehat{Q} = [\widehat{Q}_1^+ \ \widehat{Q}_1^- \ \ldots \widehat{Q}_k^+ \ \widehat{Q}_k^-]$ la matriz $n \times r$ cuyas columnas son las bases de todos los subespacios invariantes de A que han sido computados por el Algoritmo 2 `(SYSVD0)`*. Entonces existe una matriz $\bar{Q}$ de dimensión $n \times r$ con columnas exactamente ortonormales tal que*

$$\widehat{Q} = \bar{Q} + H \quad with \quad \|H\|_F = O(\epsilon). \tag{72}$$

Es necesario realizar algunos comentarios en relación al Teorema 3.8:

1. El Teorema 3.8 se aplica únicamente a los vectores propios correspondientes a valores propios no nulos. Si la matriz original A es singular con rango $r < n$, alguno de sus valores propios es nulo, y se parte de una descomposición SVD completa computada con pequeños errores multiplicativos de orden $O(\kappa\,\epsilon)$, entonces la base ortonormal del núcleo de A ($KerA$) que se proporciona en los $n - r$ vectores singulares derechos está computada con un error de orden, $O(\kappa\,\epsilon)$, ya que tanto los gaps relativos de valores singulares como los de valores propios son iguales a 1. Si se proporciona una descompresión SVD en forma

reducida, entonces también se pueden obtener vectores propios precisos correspondiente a los valores propios nulos como las $n - r$ ultimas columnas de factor ortogonal en una descomposición QR completa de las vectores singulares derechos.

2. Nótese que la cota dada en la expresión (71) del Teorema 3.8 son de hecho dos cotas independientes: una para los vectores propios correspondientes a valores propios positivos y otra para los negativos dentro de la agrupación $\widehat{\Sigma}_i$.

3. Los resultados del Teorema 3.8 serán validos siempre que se cumpla

$$\frac{\kappa\, O(\epsilon)}{relgap(\widehat{\Sigma}_i)} < 1 \tag{73}$$

3.3. Un ejemplo ilustrativo[3]

Creemos necesario presentar un ejemplo práctico para ver en un caso concreto cómo resuelve la rutina `SYSVD` la descomposición espectral de una matriz simétrica. Este mismo ejemplo será utilizado en la siguiente sección para describir las modificaciones que se han realizado del planteamiento original de la rutina. Por estas razones recomendamos su lectura para poder asimilar lo que hemos visto de la rutina y lo que resta por ver.

Vamos a seguir el esquema la rutina SYSV0 tal y como se presentó al comienzo de la sección. Partimos de la descomposición SVD de una matriz simétrica $A \in \mathbb{R}^{11\times 11}$ que se lleva a cabo en el primer paso del algoritmo SSVD por una rutina externa:

$$A = U\Sigma V^T$$

donde los valores singulares están contenidos en la diagonal de la matriz $\Sigma \in \mathbb{R}^{11\times 11}$ y los vectores singulares derechos e izquierdos, $U, V \in \mathbb{R}^{11\times 11}$ respectivamente. El objetivo es obtener los valores y los vectores propios de la matriz A.

[3] Este ejemplo se usará como referencia en varias secciones de esta memoria. Para destacar su presencia en el texto hemos usado el color azul cuando lo tratemos.

Los valores singulares σ_i de la matriz A ordenados decrecientemente son los siguientes:

σ_1	102,1688
σ_2	101,1062
σ_3	100,0317
σ_4	100,0283
σ_5	99,96551
σ_6	10,01341
σ_7	9,984753
σ_8	1,001387
σ_9	1,000222
σ_{10}	1,000047
σ_{11}	0,9996556

Para simplificar el seguimiento de este ejemplo, los valores singulares se han distribuido en tres escalas, $1, 10$ y 100. Pasamos a resolver la descomposición espectral de la matriz A siguiendo los tres pasos que se presentaban el Algoritmo 2 `(SYSVD0)`.

Primera selección de agrupaciones: Algoritmo 2.1

En primer lugar se determinarán los grupos numéricamente iguales de valores singulares: las agrupaciones. Dos valores singulares consecutivos se considerará que forman parte de la misma agrupación si su separación desde el punto de vista relativo es menor que una determinada tolerancia tol:

$$\frac{|\sigma_i - \sigma_{i+1}|}{|\sigma_i|} \leq tol$$

Para este caso concreto se ha tomado una tolerancia $tol = 10^{-4}$.

Después de realizar la primera selección de agrupaciones para nuestro ejemplo, hemos unido en una única agrupación los valores singulares σ_3 y σ_4 (ver Figura 1). Debido a esta unión, el número de agrupaciones de valores singulares es 10 de ahora en adelante.

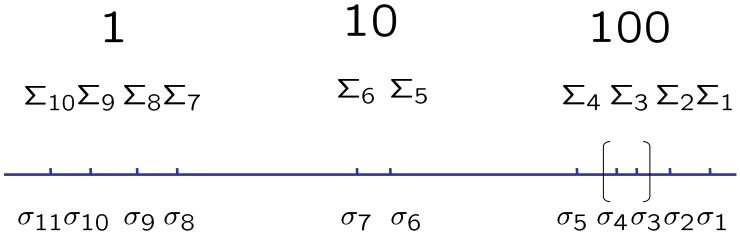

Figura 1: Distribución de los Valores Singulares σ_i y las agrupaciones Σ_i de la matriz A.

Obtención de los valores propios: Algoritmo 2.2

Se tratará ahora de obtener los valores propios de la matriz A. Debido a que A es un matriz simétrica, los valores singulares son los módulos de los valores propios. Conocido el módulo de los valores propios, el cálculo resulta bastante sencillo puesto que nos limitaremos a obtener su signo.

Para los valores singulares multiplicidad sencilla, agrupaciones de dimensión igual a 1, el signo del valor propio correspondiente lo obtendremos como el producto escalar de los vectores singulares izquierdos y derechos, u_i y v_i, correspondientes. Por ser vectores singulares de un mismo valor singular son paralelos, su producto escalar será 1 ó -1, precisamente el signo que estamos buscando.

Para la única agrupación con dimensión mayor que 1, la agrupación Σ_3, tal como se describe en la sección §3.1, calcularemos la submatriz $\Delta_3 = V_3^T U_3$ como el producto de los vectores singulares izquierdos y derechos correspondientes a σ_3. Esta submatriz Δ_3 simétrica y ortogonal tendrá una dimensión 2, igual al tamaño de la agrupación de valores singulares Σ_3. En este caso

$$V_3^T U_3 = -I_{2\times 2}$$

En el caso general será necesario diagonalizar la submatriz $V_3^T U_3$ para obtener los valores y vectores propios de A.

$$\Delta_3 = V_3^T U_3 = W_3 J_3 W_3^T$$

Los valores singulares J_3 serán 1 o -1 por tratarse de una matriz simétrica y ortogonal (Corolario 2.20), precisamente los signos de los valores propios de la matriz A que estamos buscando y los vectores propios W_i serán necesarios para obtener los vectores propios de la matriz A como veremos en el siguiente apartado.

En este caso al ser la matriz Δ_3 diagonal, lo que ocurrirá siempre que todos los valores propios en una misma agrupación tengan el mismo signo[4], por lo que obviamente no será necesario diagonalizarla, entonces los signos de los valores propios de A se encuentran directamente en su diagonal principal, los dos valores propios son negativos.

En el código finalmente implementado en FORTRAN, los valores propios se obtienen sin necesidad de diagonalizar la matriz $\Delta_3 = V_3^T U_3$. Así si no se quieren obtener los vectores propios se realizan menos operaciones. Esto se hace mediante fórmula explicada en la sección §3.1:

$$m_i^{\pm} = \frac{m_i \pm \operatorname{trace}(V_i^T U_i)}{2}$$

A través de la dimensión y la traza de la matriz Δ_3 es posible obtener los signos de los valores propios. En la agrupación Σ_3 están contenidos m_3^+ valores propios positivos y m_3^- valores propios negativos.

[4] En general las matrices Δ_i son simétricas y ortogonales no diagonales.

$$m_3^+ = \frac{m_3 \ + \mathrm{trace}(V_3^T U_3)}{2} = \frac{(2+(-2))}{2} = 0$$
$$m_3^- = \frac{m_3 \ - \mathrm{trace}(V_3^T U_3)}{2} = \frac{(2-(-2))}{2} = 2$$

Obtenemos entonces los valores propios de la matriz A "signando"los valores singulares. Tanto si la agrupación de valores singulares a la que pertenece tiene dimensión 1 como si no ocurre esto, el cálculo de los valores propios con Alta Precisión Relativa no presenta dificultad (ver Figura 2).

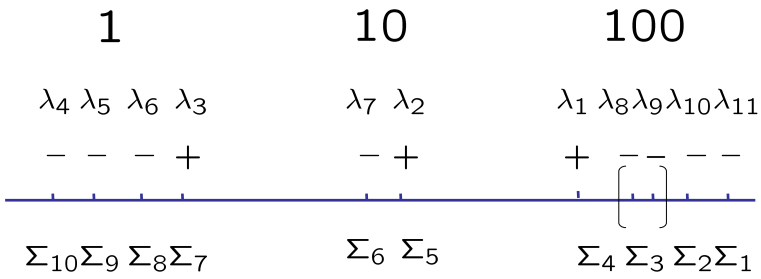

Figura 2: Distribución de los Valores Propios λ_i de la matriz A. La ordenación es $\lambda_i \geq \lambda_{i+1}$.

Obtención de los vectores propios: Algoritmo 2.3

Al igual que en la obtención de los valores propios, la mecánica de cálculo para obtener los vectores propios se ve afectada por la presencia de agrupaciones de valores singulares de dimensión mayor a 1.

Para los valores propios cuyos valores singulares tengan multiplicidad igual a la unidad, obtendremos de una manera automática los vectores propios ya que estos serán o bien los vectores singulares izquierdos o los derechos U y V. Elegiremos arbitrariamente como vectores propios los vectores derechos V. De esta manera para nuestro caso particular, sin necesidad de hacer cálculo alguno, obtendremos todos los vectores propios salvo los correspondientes a los dos valores propios contenidos en la agrupación Σ_3 de dimensión 2.

Para obtener los vectores propios restantes, como habíamos adelantado, en general necesitaremos diagonalizar la submatriz $\Delta_3 = V_3^T U_3$.

$$V_3^T U_3 = W_3 J_3 W_3^T$$

Necesitaremos hacer uso de los vectores W_i para obtener los vectores propios de A mediante la expresión (54) que se deducía en la sección §3.1:

$$Q_3 = V_3 W_3$$

Para nuestro caso particular $\Delta_3 = -\mathbb{I}_2$ y por tanto no es necesario diagonalizarla. Sus vectores propios W_3 serán la identidad, por lo tanto los vectores propios de la matriz A también

serán los vectores singulares. No obstante realizaremos este producto que en ese caso es trivial para ilustrar el procedimiento:

$$Q_3 = V_3 W_3 = V_3 I_{2\times 2} = V_3$$

En el caso general al ser la matriz ortogonal W_3 no diagonal, los vectores propios que se obtienen no serán los vectores singulares. El código esta implementado de tal forma que no se realizan estas operaciones innecesarias. Se detecta que los valores singulares de una agrupación son todos del mismo signo antes de diagonalizar la matriz Δ mediante la expresión (55), en este caso para calcular los vectores propios, se procede exactamente igual que si la agrupación tuviera dimensión 1, los vectores propios son directamente los vectores singulares.

De este modo la rutina `SYSVD0` resuelve la descomposición espectral de una matriz simétrica a partir de la SVD. Nótese que apenas se realizan algunas operaciones elementales, la gran mayoría del coste operacional del Algoritmo 1 (SSVD) está en el primer `paso`, la obtención de la descomposición SVD.

4. La rutina SYSVD definitiva

Si bien el problema del cálculo de los valores propios ya está completamente resuelto por la rutina SYSVD0, hay situaciones para las que las cotas de errores de los vectores propios (71) conseguidas por la rutina `SYSVD0` son mucho peores que las cotas deseables. Las cotas que se pueden considerar óptimas están gobernadas por el gap relativo de valores propios [4]:

$$\| \widehat{Q}_i^{\pm} - Q_i^{\pm} \|_F \leq \frac{\kappa\, O(\epsilon)}{relgap(\widehat{\Lambda}_i)}, \tag{74}$$

Por la naturaleza del Algoritmo 1 (SSVD), que toma como punto de partida la descomposición SVD, las cotas de errores en el cálculo de los vectores propios siempre estarán asociadas a los gaps relativos de valores singulares, o lo que es lo mismo, a los gaps relativos de módulos de los valores propios. Los gaps relativos de los valores singulares son siempre menores que los de valores propios, por lo que las cotas de errores para el cálculo de los vectores propios serán mayores. Aunque la mayoría de las veces esto no tendrá un efecto significativo, para determinadas distribuciones de valores singulares, los vectores propios obtenidos con la rutina `SYSVD0` si serán mucho peores que si éstas estuvieran asociadas a los gaps relativos de valores propios. Esto ocurrirá, por ejemplo, cuando un valor propio positivo λ_j, pudiendo tener valores propios negativos en la misma agrupación, esté rodeado de valores propios negativos que están mucho más cerca que el valor propio positivo más cercano, de modo que

$$relgap(\sigma_j) << relgap(\lambda_j). \tag{75}$$

Cuando se plantea un caso como éste, debido a que las cotas de errores de los vectores propios correspondientes a λ_j están asociadas a los gaps relativos de valores singulares (ver Teorema 3.8), la precisión en el cálculo de los vectores propios es mucho peor de lo que les correspondería. En esta sección se describen las mejoras sobre el Algoritmo 2 `(SYSVD0)` que solucionan esta problemática para finalmente presentar el algoritmo `SYSVD` definitivo.

Creemos necesario utilizar un ejemplo práctico para ver en un caso concreto cuál es el problema que se plantea en el cálculo de los vectores propios y como se soluciona. Por esta razón retomaremos el caso práctico visto en la sección §3.3 para apoyar la descripción de las modificaciones sobre el planteamiento inicial que conforman la rutina definitiva.

4.1. Precisión en los vectores propios: $relgap(\Sigma)$ vs $relgap(\Lambda)$

En la sección §3.3 teníamos una matriz simétrica $A \in \mathbb{R}^{11\times 11}$ con un distribución de valores propios y valores singulares representada en la Figura 2. En la Tabla 2 presentamos los valores propios y los errores cometidos por la rutina `SYSVD0` en simple precisión en el cálculo sus vectores propios correspondientes. También se representan los gaps relativos de valores propios y valores singulares asociados.

Tabla 2: Agrupaciones, valores singulares, valores propios, vectores propios, $relgap(\Sigma)$, $relgap(\Lambda)$ y errores en los vectores $\|q_i - \widehat{q}_i\|$.

	σ_i	λ_i		relgap(Σ_i)	relgap(λ_i)	$\|q_i - \widehat{q}_i\|$
Σ_1	σ_1	λ_{11}	-102.17	1.03×10^{-1}	1.03×10^{-1}	6.50×10^{-6}
Σ_2	σ_2	λ_{10}	-101.11	1.05×10^{-1}	1.05×10^{-1}	8.72×10^{-6}
Σ_3	σ_3	λ_9	-100.03	6.27×10^{-4}	1.07×10^{-1}	6.99×10^{-5}
	σ_4	λ_8	-100.03	6.27×10^{-4}	1.07×10^{-1}	6.99×10^{-5}
Σ_4	σ_5	λ_1	99.966	6.28×10^{-4}	0.89	6.99×10^{-5}
Σ_5	σ_6	λ_2	10.013	2.8×10^{-3}	0.9	8.74×10^{-6}
Σ_6	σ_7	λ_7	-9.9848	2.8×10^{-3}	0.89	8.69×10^{-6}
Σ_7	σ_8	λ_3	1.0014	1.16×10^{-3}	1	9.36×10^{-5}
Σ_8	σ_9	λ_6	-1.0002	1.75×10^{-4}	1.75×10^{-4}	2.84×10^{-4}
Σ_9	σ_{10}	λ_5	-1.0000	1.75×10^{-4}	1.75×10^{-4}	1.95×10^{-4}
Σ_{10}	σ_{11}	λ_4	-0.9997	3.91×10^{-4}	3.91×10^{-4}	3.22×10^{-4}

Cuando estudiamos el análisis de errores de la rutina `SYSVDO` en la sección §3.2 vimos que la cota de error en el cálculo de los vectores propios venía dada por la expresión (71).

Para simple precisión el mínimo error posible es la unidad de redondeo $\epsilon = 5,96 \times 10^{-8}$. La constante κ es de orden $O(n)$ aunque habitualmente es menor [6]. Entonces se puede esperar que $-\log_{10}(relgap(\widehat{\Sigma}_i))$ sean los dígitos que se pierden en el cálculo de los vectores propios. Puede comprobarse que esto es cierto comparando las columnas 5 y 7 de la Tabla 2. La precisión que se pierde en los vectores propios (columna 7) coincide aproximadamente con el gap relativo de valores singulares (columna 5).

El problema lo encontramos en los vectores destacados en rojo, aquí se plantea la situación que hemos comentado, $relgap(\Sigma_j) << relgap(\Lambda_j)$. Para estos los valores propios los gaps relativos de valores propios son de orden 1, mucho mayores que los gaps relativos de valores singulares. Para los vectores propios correspondientes a estos valores propios se llegan a perder 4 cifras significativas, cuando a la vista de sus relgap de valores propios, λ_3 por ejemplo, no se debería perder ninguna. Esto ocurre porque para el algoritmo `SSVD` las cotas de error están gobernados por los gaps relativos de valores singulares debido que se parte de la descomposición SVD. Si las cotas de error estuvieran gobernadas por los gaps relativos de valores propios, lo que ocurre en los algoritmos convencionales, no se debería perder apenas una cifra decimal de precisión para estos vectores.

Las mejoras que se presentan en esta sección solucionarán este problema de manera que, aunque para el algoritmo SSVD las cotas siempre estén gobernadas por los gaps relativos de valores singulares, se limitará la pérdida de precisión modificando la estructura de agrupaciones

de valores singulares.

4.2. Modificiaciones/mejoras sobre la rutina SYSVD0

Si estudiamos detenidamente el Teorema 3.8, a la vista de las cotas de errores de vectores propios que el Algoritmo 1 (SSVD) proporciona, se puede deducir un principio muy interesante que permitirá trabajar sobre la problemática de la precisión de los vectores:

Los resultados del Teorema 3.8 son independientes de cómo se seleccionen las agrupaciones de valores singulares, siempre que se obedezca (73).

Haciendo uso de este principio, es posible hacer nuevas agrupaciones de valores singulares $\{\Sigma'_i\}_{i=1}^{q}$, de manera que

$$relgap(\Sigma'_i) \lesssim relgap(\Lambda'_i)$$

y así obtener unas cotas de errores que, aunque siempre estén asociadas a los gaps relativos de valores singulares, proporcionen una precisión deseable para los vectores propios. Este principio está detrás de las modificaciones que se realizarán sobre el Algoritmo 2 `(SYSVD0)`.

El objetivo es definir unas nuevas agrupaciones de valores singulares de modo que los correspondientes vectores estén computados con un error más pequeño (71) al pertenecer a agrupaciones con un $relgap(\Sigma)$ más grande. Este incremento de los gaps relativos de los valores singulares no nos impedirá utilizar el Teorema 3.8 porque estas agrupaciones no dejarán de cumplir la condición (73).

Como veremos, no es posible resolver el problema de la precisión en los vectores propios de una sola vez, las mejoras deben integrarse en el código en tres fases. Recordamos una vez más que estas modificaciones no afectan en absoluto al cálculo de los valores propios, todas las modificaciones sobre el código se realizan una vez obtenidos estos:

- Segunda selección de agrupaciones: entre los pasos 2 y 3 del Algoritmo 2 `(SYSVD0)`, después de la obtención de los valores propios y antes del la obtención de los vectores propios, se realizará una segunda selección de agrupaciones de valores singulares. Esta nueva distribución de agrupaciones supondrá una mejora en el primer cálculo de algunos vectores propios, esta fase resultará fundamental para que en el tercer paso sea posible mejorar la precisión del resto de los vectores.
- Tercera selección de agrupaciones: Se realizará una tercera y última selección de agrupaciones de valores singulares después de la obtención de los vectores propios.
- Segundo cálculo de los vectores propios: Se calcularán nuevamente algunos vectores propios del mismo modo que se realizó en el `paso` 3 del Algoritmo 2 `(SYSVD0)` pero utilizando la tercera selección de agrupaciones, estos se combinarán adecuadamente con los vectores calculados anteriormente y se obtendrán así finalmente los vectores precisos que formarán una de base los subespacios propios de la matriz A.

El pseudocódigo que se presenta a continuación, es la versión final de la rutina `SYSVD`. La rutina definitiva integra la rutina `SYSVD0` (Algoritmos 2, 2.1, 2.2 y 2.3) y las mejoras que desarrollaremos a lo largo de esta sección, `REFINAMIENTO 1` (Algoritmo 3.1) y `REFINAMIENTO 2` (Algoritmo 3.2).

Algorithm 3 (SYSVD)
INPUT:

- Descomposición en valores singulares de la matriz simétrica $A = U\Sigma V^T$
- Parámetros: tol < 1, tolgap < 1, tolgap2 < 1.

OUTPUT:

- Valores propios $\Lambda = diag[\lambda_i]$
- Vectores propios $Q = [q_1 \dots q_n]$; $A = Q\Lambda Q^T$

1. *Realizar la primera selección de agrupaciones de valores singulares,* $\{\Sigma_i, U_i, V_i\}_{i=1}^k$, *mediante el Algoritmo* 2.1 *con tolerancia* tol.
2. *Computar los valores propios utilizando el Algoritmo* 2.2.
3. *Usar el Algoritmo* 3.1 *para unir, cuando sea necesario, algunos pares de agrupaciones para formar una distribución de agrupaciones* $\{\Sigma_i, U_i, V_i\}_{i=1}^q$ *con tolerancia* tolgap.
4. *Computar los vectores propios utilizando el Algoritmo* 2.3 *para la nueva distribución de agrupaciones*
5. *Realizar una tercera definitiva selección de agrupaciones y obtener unos nuevos vectores propios mediante el* Algoritmo 3.2 *con tolerancia* tolgap2.

4.2.1. Segunda selección de agrupaciones de valores singulares

Como se ha comentado, ocurre que en algunos casos la selección inicial de agrupaciones (60) puede ser modificada de modo que las cotas de error (71) sean menores para algunos los vectores propios. Esta segunda selección de agrupaciones se realiza después del cálculo de los valores propios y antes de calcular los vectores propios por primera vez. Algunos de los vectores propios calculados después de esta selección de agrupaciones serán mejorados, el resto serán mejorados a través de la tercera selección de agrupaciones.

La idea básica de esta segunda selección de agrupaciones no es complicada:

1. Sea por ejemplo $\widehat{\Sigma}_i = \widehat{\Lambda}_i^+$ una de las agrupaciones de valores singulares seleccionadas de acuerdo a la definición 3.4, que contenga valores propios de un solo signo (positivo, sin pérdida de generalidad). El gap relativo $relgap(\widehat{\Sigma}_i)$ puede ser mucho más pequeño, mucho peor de cara a obtener los vectores propios, que el gap relativo de valores propios de $\widehat{\Lambda}_i$. Supóngase además que la agrupación más cercana (en el sentido relativo) a $\widehat{\Sigma}_i$ sólo tiene valores propios negativos, es decir[5], $\widehat{\Sigma}_{cl(i)} = |\widehat{\Lambda}_{cl(i)}^-|$. Si se unen ambas agrupaciones para formar una nueva agrupación $\widehat{\Sigma}_i' = \widehat{\Lambda}_i^+ \cup |\widehat{\Lambda}_{cl(i)}^-|$ con un mayor relgap, la cota de error para los vectores propios (71) se mejorará **separadamente** para las bases de los *dos subespacios invariantes* asociados a $\widehat{\Lambda}_i^+$ y $\widehat{\Lambda}_{cl(i)}^-$ en la computación del algoritmo SYSVD para esa nueva agrupación. En conclusión, nada se pierde uniendo dos agrupaciones que estén totalmente signadas de manera opuesta y la cota de error (71) de la nueva agrupación

[5] $\Sigma_{cl(i)}$ denota la agrupación más cercana, en sentido relativo, a Σ_i

puede ser mejorada en la medida que el gap relativo asociado a esta sea mayor que para las agrupaciones por separado.

2. Se han de dar tres condiciones para que la unión de agrupaciones adyacentes tenga lugar.

   a) Los elementos de cada agrupación deberán ser todos del mismo signo y las dos agrupaciones deberán ser de signo opuesto. De otra manera no se podría recuperar la información de los subespacios propios del subespacio invariante suma de las agrupaciones.

   b) El gap relativo de valores singulares de una de las agrupaciones debe ser suficientemente más pequeño que el gap relativo de valores propios (esto está definido por la tolerancia tolgap, que habitualmente toma un valor de 0.5)

   c) El gap relativo de la nueva agrupación deberá ser mayor que el mínimo de los gaps relativos de las dos agrupaciones por separado. Esto hace que siempre se mejore el peor de los dos gaps relativos de las agrupaciones involucradas.

   En otros casos no se unen las agrupaciones, porque o bien no hay una mejora significativa que justifique la unión o bien no se podría recuperar la información de los subespacios propios resultando unos vectores propios totalmente incorrectos.

Esta es la idea que esta detrás del `paso` 3 del la rutina `SYSVD`: la segunda selección de agrupaciones. Presentamos a continuación (en la siguiente página) el código de forma esquemática correspondiente a esta parte del algoritmo:

Algorithm 3.1 (REFINAMIENTO 1)

INPUT:

- Agrupaciones obtenidas por SYSVD0: $\{\widehat{\Sigma}_i = \widehat{\Lambda}_i^+ \cup |\widehat{\Lambda}_i^-|\}_{i=1}^k$
- tolgap: Tolerancia para decidir sobre que agrupaciones actuar.

OUTPUT:

- Nuevas agrupaciones $\{\widehat{\Sigma}_i = \widetilde{\Lambda}_i^+, \cup|\widetilde{\Lambda}_i^-|\}_{i=1}^q$

1. for $j = 1 : r$ % *Para las* r *agrupaciones cuyos elementos sean todos positivos (resp negativos)* $\widehat{\Sigma}_i = \widehat{\Lambda}_i^+$ *y sus agrupaciones más cercanas tengan todos sus elementos negativos (resp positivos)* $\widehat{\Sigma}_{i+1} = \widehat{\Lambda}_{i+1}^-$
2. if $relgap(\widehat{\Sigma}_i)/relgap(\widehat{\Lambda}_i^+) <$ tolgap then % *Si el cociente es menor que una determinada tolerancia* tolgap *podemos considerar unir las dos agrupaciones* $\widehat{\Sigma}_i$ y $\widehat{\Sigma}_{i+1}$ *en una nueva agrupación.*
3. $relgap(\widehat{\Sigma}_i \cup \widehat{\Sigma}_{i+1})$ % *Obtenemos el gap relativo de valores singulares de la nueva agrupación unión de dos agrupaciones consecutivas*
4. if $relgap(\Sigma_{i_c} \cup \Sigma_{i_c+1}) > \min\{relgap(\Sigma_{i_c}), relgap(\Sigma_{i_c+1})\}$ then % *Si el mínimo de los gaps relativos de las agrupaciones por separado* $\widehat{\Sigma}_i$ y $\widehat{\Sigma}_{i+1}$ *es menor que el de la nueva agrupación*
5. $\widehat{\Sigma}_i = \widehat{\Sigma}_i \cup \widehat{\Sigma}_{i+1}$ %*Procedemos a la unión, actualizando la información de todas las agrupaciones. Pasamos a considerar el siguientes candidato.*
6. end if
7. end if
8. end for

Apliquemos esta segunda selección sobre el caso práctico para ver cómo se modifica la distribución de agrupaciones (Figura 3).

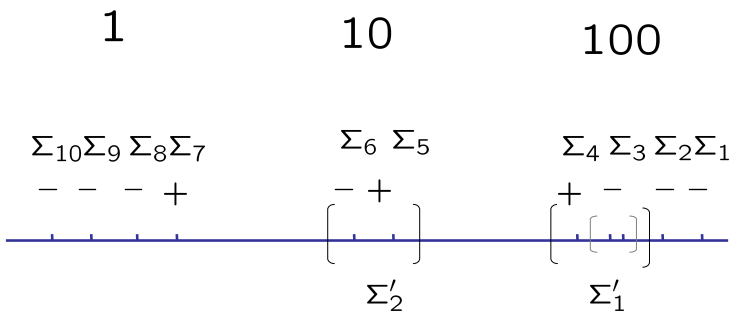

Figura 3: Distribución de los Valores Singulares σ_i tras la segunda selección de agrupaciones.

Como resultado de la segunda selección de agrupaciones de valores singulares se han producido dos uniones[6], Σ'_1 y Σ'_2 . Los agrupaciones Σ_3 y Σ_4 se han unido en una sola agrupación Σ'_1, lo mismo ha ocurrido para las agrupaciones Σ_5 y Σ_6, unidas en la agrupación Σ'_2. Para estas agrupaciones se han dado las condiciones necesarias para que se produzca la unión:

- Las agrupaciones involucradas sólo contienen valores propios de un signo.
- Se han unido agrupaciones de signos opuestos.
- El gap relativo de la nueva agrupación es mayor que el mínimo de los gaps relativos de las dos agrupaciones por separado.

Tras esta segunda selección de agrupaciones siguiendo los pasos del Algoritmo 3 (`SYSVD`) se procede a calcular los vectores propios. En la Tabla 3 se presenta cual es la situación de los vectores propios en este punto de la rutina.

La nueva selección de agrupaciones modifica los gaps relativos de valores singulares y por tanto la precisión en el cálculo de los vectores. Se han dado diferentes situaciones:

Los gaps relativos de valores singulares de Σ_5 y Σ_6, a los que les corresponden los valores propios λ_2 y λ_7 respectivamente, ahora pertenecen a la nueva agrupación Σ'_2. Para esta nueva agrupación los gaps relativos de valores singulares y los de propios se equiparan y por consiguiente, sus vectores propios se obtienen con la precisión deseada (en verde).

Para la nueva agrupación Σ'_1 que contiene a Σ_3 y Σ_4, se dan dos situaciones:

- Para Σ_3 ocurre lo mismo que en caso anterior, la nueva distribución hace que los gaps relativos de valores singulares y propios se equiparan y se obtiene la precisión deseada para los vectores propios correspondientes (en verde).

[6]Para mayor claridad en la descripción estas agrupaciones has sido identificadas con índices independientes, en el código no hay distinción entre estas las agrupaciones creadas en la segunda selección de agrupaciones y en la primera, tras cada unión el numero de agrupaciones se reduce y los índices necesarios son corregidos. Estas nuevas agrupaciones quedan integradas de forma consecutiva entre las demás.

Tabla 3: Valores propios, vectores propios, relgap Σ, relgap λ y error en los vectores

		σ_i	λ_i		relgap(Σ_i)	relgap(λ_i)	$\|q_i - \widehat{q}_i\|$
	Σ_1	σ_1	λ_{11}	-102.17	1.03×10^{-1}	1.03×10^{-1}	6.50×10^{-6}
	Σ_2	σ_2	λ_{10}	-101.11	1.05×10^{-1}	1.05×10^{-1}	8.72×10^{-6}
Σ_1'	Σ_3	σ_3	λ_9	-100.03	1.07×10^{-1}	1.07×10^{-1}	8.26×10^{-6}
		σ_4	λ_8	-100.03	1.07×10^{-1}	1.07×10^{-1}	8.26×10^{-6}
	Σ_4	σ_5	λ_1	99.966	1.07×10^{-1}	0.89	5.69×10^{-6}
Σ_2'	Σ_5	σ_6	λ_2	10.013	0.89	0.9	2.36×10^{-7}
	Σ_6	σ_7	λ_7	-9.9848	0.89	0.89	1.97×10^{-7}
	Σ_7	σ_8	λ_3	1.0014	1.16×10^{-3}	1	9.36×10^{-5}
	Σ_8	σ_9	λ_6	-1.0002	1.75×10^{-4}	1.75×10^{-4}	2.84×10^{-4}
	Σ_9	σ_{10}	λ_5	-1.0000	1.75×10^{-4}	1.75×10^{-4}	1.95×10^{-4}
	Σ_{10}	σ_{11}	λ_4	-0.9997	3.91×10^{-4}	3.91×10^{-4}	3.22×10^{-4}

- Para Σ_4, sin embargo, se da una situación diferente, aunque el gap relativo de valores singulares ha aumentado, y por consiguiente se ha mejorado la precisión en el vector propio correspondiente, a la vista del gap relativo de valores propios es posible mejorar más (en amarillo). Antes el gap relativo de Σ_4 lo producía Σ_3 por ser la agrupación más cercana, al agrupar Σ_3 y Σ_4 el gap relativo lo produce Σ_2, pero el que corresponde es de orden unidad, producido por el valor propio λ_2. Este vector propio se mejorará definitivamente en el segundo refinamiento.

El otro vector que anteriormente se calculaba con baja precisión, el correspondiente a λ_3, Σ_7, debido que la nueva selección de agrupaciones no modifica el gap relativo valores singulares de Σ_7, no se produce mejora (en rojo). Las agrupaciones Σ_7 y Σ_8 no se han unido puesto que a pesar de cumplir las dos de las tres condiciones necesarias para ello, el gap relativo resultante de la nueva agrupación no sería mayor que el de las agrupaciones por separado por la cercanía de Σ_8. Este vector propio será mejorado totalmente en el segundo y definitivo refinamiento.

4.2.2. Tercera selección de agrupaciones de valores singulares

Después de ejecutar la primera y segunda selección de agrupaciones con las respectivas tolerancias `tol` y `tolgap`, tenemos una distribución de agrupaciones de valores singulares $\{\widehat{\Sigma}_i = \widehat{\Lambda}_i^+ \cup |\widehat{\Lambda}_i^-|\}_{i=1}^q$, y sus respectivas agrupaciones de valores propios.

Para cada una de estas agrupaciones de valores singulares, en el `paso` 4 del Algoritmo 3 se han calculado los vectores propios correspondientes, las matrices $\widehat{Q}_i = [\widehat{Q}_i^+ \ \widehat{Q}_i^-]$, $i = 1, \ldots, q$, con los errores dados en (71).

Dada la actual distribución de agrupaciones de valores singulares y valores propios se pueden presentar dos situaciones *para cada agrupación de valores propios, positiva o negativa:* $\widehat{\Lambda}_i^*$, $* = +, -$.

1. $relgap(\widehat{\Lambda}_j^*) \gtrsim relgap(\widehat{\Sigma}_j)$. Situación para la cual ya se ha alcanzado una precisión en los vectores adecuada (74). Las cotas de error para los vectores pese a estar asociadas a los gaps relativos valores singulares, como estos son del orden de los gaps relativos de valores propios, son las mejores posibles.

   En nuestro ejemplo práctico este es el caso de las agrupaciones de valores propios $\{\lambda_{11}\}$, $\{\lambda_{10}\}$, $\{\lambda_9, \lambda_8\}$, $\{\lambda_2\}$, $\{\lambda_7\}$,$\{\lambda_6\}$, $\{\lambda_5\}$ y $\{\lambda_4\}$, es decir, para todas menos para $\{\lambda_1\}$ y $\{\lambda_3\}$ (ver Tabla 3).

2. $relgap(\widehat{\Lambda}_j^*) >> relgap(\widehat{\Sigma}_j)$. Ocurre cuando, por ejemplo, una agrupación de valores propios positivos $\widehat{\Lambda}_j^+$ esta rodeada de agrupaciones de valores propios exclusivamente negativos que están mucho más cerca que el valor propio positivo más cercano. Dada esta situación, las cotas de error que proporciona (71) son mucho peores de lo deseable.

   En nuestro ejemplo práctico este es el caso de las agrupaciones de valores propios $\{\lambda_1\}$ y $\{\lambda_3\}$ (ver Tabla 3). Estos dos casos son agrupaciones de valores propios de un sólo valor propio positivo que están rodeados de valores propios negativos.

Se puede resolver la segunda situación mediante una tercera y definitiva selección de agrupaciones. Supóngase que se tiene una agrupación $\widehat{\Sigma}_j = \widehat{\Lambda}_j^+ \cup |\widehat{\Lambda}_j^-|$, tal que $\widehat{\Lambda}_j^+$ no está vacío[7], que su agrupación más cercana, en sentido relativo, $\widehat{\Sigma}_{cl(j)}$, es negativa y que

$$relgap(\widehat{\Sigma}_j) < \texttt{tolgap2} \cdot relgap(\widehat{\Lambda}_j^+). \tag{76}$$

para alguna constante $\texttt{tolgap2} < 1$. Se demuestra en [5] que es posible formar una nueva agrupación, formada alrededor de $\widehat{\Sigma}_j$,

$$\widehat{\Sigma}_j'' = \cup_{k=r'}^{l'} \widehat{\Sigma}_k \equiv |\widehat{\Lambda}_j''^-| \cup \widehat{\Lambda}_j^+, \tag{77}$$

con $\widehat{\Lambda}_j''^- = \cup_{k=r'}^{l'} \widehat{\Lambda}_k^-$, que es la unión de las agrupaciones negativas $\widehat{\Sigma}_k = \widehat{\Lambda}_k^-,\ k = r' : l',\ k \neq j$, y $\widehat{\Sigma}_j$, que satisface

$$relgap(\widehat{\Sigma}_j'') = O\left(\frac{1}{n}\right) relgap(\widehat{\Lambda}_j^+). \tag{78}$$

De modo que para la nueva agrupación $\widehat{\Sigma}_j''$, la cota de error de los vectores propios gobernada por el gap relativo de valores singulares (71) se puede llegar comportar como si estuviera gobernada por los vectores propios (74), salvo un factor de que a lo sumo es de orden $O(n)$ (y habitualmente es menor).

Este proceso (y todo el algoritmo) es demostrado de manera formal en [5], de acuerdo con el objetivo de este proyecto presentamos aquí una breve descripción del procedimiento.

En lo sucesivo, y para mayor claridad, nos referimos a la nueva agrupación $\widehat{\Sigma}_j''$ como macro-agrupación, agrupación de agrupaciones.

[7] Vamos a suponer en esta argumentación, sin pérdida de generalidad, que la agrupación de valores propios es positiva, al igual que ocurre en el ejemplo práctico.

Volveremos momentáneamente al caso práctico que hemos ido resolviendo progresivamente en las últimas secciones. Estudiemos cómo se forman estas macroagrupaciones en la distribución de valores singulares y valores propios que hemos visto anteriormente (Figura 3):

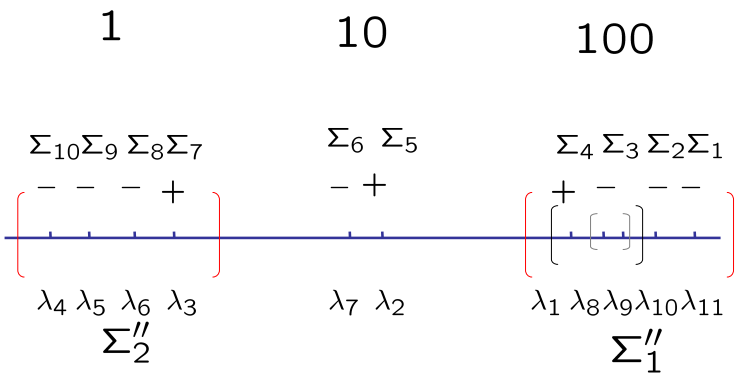

Figura 4: Tercera selección de agrupaciones de valores singulares.

Debido a que, por la naturaleza de la rutina `SYSVD`, las cotas de errores para los vectores propios están gobernadas por el gap relativo de valores singulares y no por los gaps relativos de valores propios, la precisión en los vectores q_1 y q_3, correspondientes, respectivamente, a los valores propios λ_1 y λ_3, son peores de lo que les correspondería. Esto ocurre porque la agrupaciónes positiva más cercana para ambas λ_2 produciría un **gap relativo de valores propios** de orden $O(1)$, mucho mayor que el **gap relativo de valores singulares.**

Como hemos comentado, dada esta situación, es posible crear dos macroagrupacioes para resolver el problema de en los vectores propios correspondientes a λ_1 y λ_3. La primera macroagrupación, a la que llamaremos Σ_1'', se formará con el valor propio negativo λ_1 y las agrupaciones positivas que lo rodean (Σ_1, Σ_2, Σ_3), es decir:

$$\Sigma_1'' = \{\Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4\} = \{\lambda_1, |\lambda_{11}|, |\lambda_{10}|, |\lambda_9|, |\lambda_8\}|.$$

De la misma forma, la segunda macroagrupación, a la que llamaremos Σ_2'', se formará con el valor propio negativo λ_3 y las agrupaciones positivas que lo rodean (Σ_8, Σ_9, Σ_{10}), es decir:

$$\Sigma_2'' = \{\Sigma_7 \cup \Sigma_8 \cup \Sigma_9 \cup \Sigma_{10}\} = \{\lambda_3, |\lambda_6|, |\lambda_5|, |\lambda_4|\}$$

Los gaps relativos de esta nuevas macro agrupaciones, Σ_1'' y Σ_2'', (paréntesis rojos en la Figura 4) satisfacen (78).

Dejamos por ahora el caso práctico para continuar con la descripción del procedimiento.

Para las nuevas macroagrupaciones creadas con el procedimiento esbozado antes, ahora se puede ejecutar el Algoritmo 2.3 para obtener una buena base del subespacio invariante suma directa de los subespacios propios de cada uno de las agrupaciones de valores propios que forman $\widehat{\Sigma}_j''$. Estos vectores son

$$\widehat{Z} = [\widehat{Z}^+,\ \widehat{Z}^-]$$

donde $\widehat{Z}^+$ y $\widehat{Z}^-$ son los vectores computados de las bases de los subespacios invariantes $S(\widehat{\Lambda}_j^+)$ y $S(\widehat{\Lambda}_j''^-)$, respectivamente, que se corresponden con los valores propios positivos y negativos de A que se encuentran en la macroagrupación $\widehat{\Sigma}_j''$.

Tenemos que:

$$\| \widehat{Z}^+ - Q_j^+ \|_F \leq \frac{\kappa\, O(\epsilon)}{relgap(\widehat{\Sigma}_j'')} \approx \frac{n\, \kappa\, O(\epsilon)}{relgap(\widehat{\Lambda}_j^+)} \tag{79}$$

$$\| \widehat{Z}^- - Q_j''^- \|_F \leq \frac{\kappa\, O(\epsilon)}{relgap(\widehat{\Sigma}_j'')} \approx \frac{\kappa\, O(\epsilon)}{relgap(\widehat{\Lambda}_j''^-)}. \tag{80}$$

donde las columnas de Q_j^+ y $Q_j''^-$ son una base ortonormal de los subespacios invariantes $S(\widehat{\Lambda}_j^+)$ y $S(\widehat{\Lambda}_j''^-)$.

La cota (79) es precisamente la que se quiere obtener para los vectores propios del subespacio $S(\widehat{\Lambda}_j^+)$, los errores en los vectores propios de $\widehat{\Lambda}_j^+$ están gobernados por los gaps relativos de valores propios.

No obstante se plantea un problema diferente para el subespacio $S(\widehat{\Lambda}_j''^-)$. Aunque la cota (80) nos dicen que los vectores $\widehat{Z}^-$ son una buena base del subespacio invariante $S(\widehat{\Lambda}_j''^-)$, la información de los subespacios propios de cada una de las agrupaciones contenidas en la macroagrupación $\widehat{\Lambda}_j^-$ está mezclada y no es accesible, por lo que para obtener unos vectores propios precisos necesitaremos separar la información que corresponda a cada uno de los valores propios.

Llegado este punto conviene hacer un resumen de la situación. Hemos computado dos colecciones de vectores propios cuyos valores propios están en la macroagrupación

$$\widehat{\Sigma}_j'' = \widehat{\Lambda}_{r':j-1}^- \cup \widehat{\Lambda}_j^+ \cup \widehat{\Lambda}_j^- \cup \widehat{\Lambda}_{j+1:l'}^- = \widehat{\Lambda}_j''^- \cup |\widehat{\Lambda}_j^+|$$

1. Los vectores propios computados en el `paso` 4 de la rutina `SYSVD`: $\widehat{Q}_j^+$ y $\widehat{Q}_k^-$, $k = r' : l'$.
2. Los vectores propios computados por el procedimiento de refinamiento descrito en esta sección: $\widehat{Z}^+$ y $\widehat{Z}^-$.

No obstante ninguno de los dos conjuntos de vectores son los vectores propios precisos que estamos buscando:

1. Los vectores $\widehat{Z}^+$ y $\widehat{Z}^-$ obtenidos por el nuevo procedimiento son:
   a) Buenas bases de $S(\widehat{\Lambda}_j^+)$
   b) Malas bases $S(\widehat{\Lambda}_k^-)$, $k = r' : l'$
2. Y por el contrario los vectores $\widehat{Q}_j^+$ y $\widehat{Q}_k^-$, $k = r' : l'$ computados por el Algoritmo 2 `(SYSVDO)` son:
   a) Malas bases de $S(\widehat{\Lambda}_j^+)$
   b) Buenas bases de $S(\widehat{\Lambda}_k^-)$, $k = r' : l'$

Para conseguir las bases adecuadas de $S(\widehat{\Lambda}_j^+) \bigoplus S(\widehat{\Lambda}_j''^-)$ no podemos tomar directamente los vectores 1.(a) y 2.(b), porque al ser obtenidos por procedimientos distintos no podría garantizarse la ortogonalidad de los vectores propios. Sin embargo, es posible encontrar un algoritmo que proporciona las bases deseadas a partir de los dos conjuntos de vectores obtenidos conservando la ortogonalidad.

Lo que el algoritmo en cuestión debe hacer tomar como vectores obtenidos en 1.(a) como base del subespacio $S(\widehat{\Lambda}_j^+)$ y para obtener la bases adecuadas para los subespacios $S(\widehat{\Lambda}_k^-)$, $k = r' : l'$, se deben proyectar ortogonalmente los vectores propios obtenidos en 2.(b) sobre la base del subespacio invariante obtenido en 1.(b). De esta manera, el algoritmo tomará como bases de los subespacios invariantes $S(\widehat{\Lambda}_k^-)$, $k = r' : l'$, las columnas de la matriz:

$$\widehat{Q}''^- = [\widehat{Q}''^-_{r':j-1}, \widehat{Q}''^-_j, \widehat{Q}''^-_{j+1:l'}] = \widehat{Z}^- P, \tag{81}$$

donde P es el factor ortogonal Q de la descomposición QR de

$$\widehat{Z}^{-T} (\widehat{Q}^- \Pi_{\widehat{Q}}) = PR \tag{82}$$

$\Pi_{\widehat{Q}}$ es la permutación por bloques que reordena los bloques de $\widehat{Q}$ con con orden decreciente de sus respectivos relgaps de valores singulares. Esto es

$$\widehat{Q}^- \Pi_{\widehat{Q}} = [\widehat{Q}^-_{\Pi(1)}, \widehat{Q}^-_{\Pi(2)}, \dots \widehat{Q}^-_{\Pi(i)}, \widehat{Q}^-_{\Pi(i+1)}, \dots] \tag{83}$$

Los vectores propios negativos de la macroagrupación se ordenan de esta forma antes de hacer la descomposición QR.

$$relgap(\widehat{\Sigma}_{\Pi(i)}) \geq relgap(\widehat{\Sigma}_{\Pi(i+1)}) \tag{84}$$

para que las bases de los vectores propios resultantes no queden todas calculadas con el mayor error de todos (ver [5]).

Las ideas expuestas arriba para obtener los definitivos vectores propios se están implementadas en el siguiente algoritmo (ver página siguiente):

Algorithm 3.2 (REFINAMIENTO 2)

INPUT:

- Agrupaciones después de ejecutar el paso 3 del Algoritmo 3 (SYSVD): $\{\Sigma_i = \Lambda_i^+ \cup |\Lambda_i^-|\}_{i=1}^q$
- Vectores propios calculadas por SYSVD0: $\widehat{Q} = [\widehat{Q}_1^+ \ \widehat{Q}_1^- \ \ldots \widehat{Q}_q^+ \ \widehat{Q}_q^-]$
- tolgap2

OUTPUT:

- Nuevos vectores propios: $\widehat{Q} = [\widehat{Q}_1^+ \ \widehat{Q}_1^- \ \ldots \widehat{Q}_q^+ \ \widehat{Q}_q^-]$

1. for $j = q : -1 : 1$ *%para cada agrupación elegimos su signo de modo que* $\pm = arg(\text{máx}\{relgap(\Lambda_j^+), relgap(\Lambda_j^-)\})$
2. *% en lo sucesivo supondremos que el signo de la agrupación es +*
3. if $relgap(\Sigma_j)/relgap(\Lambda_j^+) <$ tolgap2 then
4. find l', r' *% las fronteras de modo la macroagrupación* Σ_j'' *satisfaga* (78)

$$\Sigma_j'' = [\Lambda^-_{r':j-1},\ \Lambda_j^-, \Lambda_j^+,\ \Lambda^-_{j+1:l'}]$$

5. ejecutar paso 4 *del Algoritmo 2* (SYSVD0) *sobre* Σ_j'' *para obtener* $\widehat{Z}^+$ y $\widehat{Z}^-$
6. obtener P, *% como el factor* Q *de la descomposición* QR *de* $\widehat{Z}^{-T}(\widehat{Q}^-\Pi_{\widehat{Q}}) = PR$
7. obtener $[\widehat{Q}'^-_{r':j-1}, \widehat{Q}'^-_j, \widehat{Q}'^-_{j+1:l'}] = \widehat{Z}^-(P\Pi^T_{\widehat{Q}})$
8. sustituir $\widehat{Q}^-_{r':l'} = \widehat{Q}'^-_{r':jl}$
9. sustituir $\widehat{Q}_j^+ = \widehat{Z}^+$
10. endif
11. endfor

Nótese que el algoritmo 3.2 sólo se aplica a las agrupaciones para la que hay valores propios positivos o negativos que tengan $relgap(\Sigma_i)$ suficientemente más pequeño que $relgap(\Lambda_i^*)$. El criterio para considerar que $relgap(\Sigma_i)$ sea suficientemente más pequeño que $relgap(\Lambda_i^*)$ es que

$$\frac{relgap(\Sigma_j)}{relgap(\Lambda_j^+)} < \texttt{tolgap2}.$$

El parámetro de control tolgap2 conviene fijarlo al valor tolgap2$= 1/(2\,n)$ debido a (78).

Volvamos nuevamente al ejemplo práctico para continuar aplicando este segundo procedimiento de refinado tal y como lo hemos descrito. En la Figura 3 y en la Tabla 3 está reflejada la situación en el cálculo de los vectores propios después de aplicar el primer refinamiento, `paso` 4 de la rutina `SYSVD`.

Los valores propios $\lambda_1 = 99.966$ y $\lambda_3 = 1.0014$ se encuentran rodeados de agrupaciones de valores singulares de signo opuesto que producen unos gaps relativos mucho más pequeños. Por esta razón, sus vectores propios han sido obtenidos con una precisión peor que la que le correspondería. En este caso, tanto para λ_1 como para λ_3, los gaps relativos de valores propios están producidos por $\lambda_2 = 10.013$, el único valor propio positivo restante. Ambos vectores propios serán corregidos en este procedimiento.

Hemos visto que después de aplicar la tercera selección de agrupaciones se formaban dos macroagrupaciones Σ_1'' y Σ_2'' alrededor de los dos valores propios positivos cuyos vectores propios nos disponíamos a mejorar (Figura 4).

Las macroagrupaciones Σ_1'' y Σ_2'' satisfacen (78), de manera que sus gaps relativos de valores singulares son comparables a los gaps relativos de valores propios, es este caso son prácticamente iguales.

Para estas macroagrupaciones se obtiene la segunda colección de vectores $(\widehat{Z}_1^+, \widehat{Z}_1^-)$ y $(\widehat{Z}_2^+, \widehat{Z}_2^-)$ y se plantea la situación descrita anteriormente:

- Los vectores $\widehat{Z}_1^+$ y $\widehat{Z}_2^+$ son vectores precisos correspondientes a los valores λ_1 y λ_3 que estamos buscando, q_1 y q_3.
- Para los valores propios positivos contenidos en las macroagrupaciones, los vectores Z_1^- y Z_2^- no sirven porque, a pesar de ser buenas bases del subespacio invariante que genera la parte negativa de la macroagrupación, la información de los subespacios propios correspondientes a los valores propios negativos se encuentra mezclada. No podemos quedarnos con los vectores propios precisos obtenidos en el `paso` 4 por problemas de ortogonalidad. Procedemos a proyectar los vectores propios precisos que ya hemos calculado en el `paso` 4 del Algoritmo 3 sobre los vectores Z_1^- y Z_2^- para obtener los vectores propios correspondientes a los valores propios negativos de las macroagrupaciones.

En la Tabla 4 se presentan los resultados tras el segundo y definitivo refinamiento. Finalmente todos los vectores propios que hasta ahora no se habían podido obtener de manera precisa, correspondientes a los valores propios λ_1 y λ_3, han sido refinados y su precisión se corresponde con los gap relativos de las macroagrupaciones Σ_1'' y Σ_2'', comparables a las de valores propios.

Es importante comprender la necesidad del primer refinamiento ya que como hemos visto no es posible mejorar todos los vectores directamente en este último procedimiento. ¿Por qué es necesario el primer refinamiento?

- Es necesario contar con unos vectores propios precisos antes del segundo refinamiento para proyectar los vectores Q sobre los Z y así poder separar las bases $S(\widehat{\Lambda}_k^-)$, $k = r' : l'$ de cada subespacio propio de forma que estas sean precisas.
- Para un caso como nuestro ejemplo práctico, el primer refinamiento puede corregir totalmente la precisión de los vectores $\lambda_8, \lambda_9 \in \Sigma_3$ mediante la formación de Σ_1'', algo que no se puede conseguir mediante el segundo refinamiento exclusivamente.

Tabla 4: Valores propios, vectores propios, relgap Σ, relgap λ y error en los vectores

		σ_i	λ_i		relgap(Σ_i)	relgap(λ_i)	$\|q_i - \widehat{q}_i\|$
Σ_1''	Σ_1	σ_1	λ_{11}	-102.17	1.03×10^{-1}	1.03×10^{-1}	6.50×10^{-6}
	Σ_2	σ_2	λ_{10}	-101.11	1.05×10^{-1}	1.05×10^{-1}	8.72×10^{-6}
	Σ_1'	σ_3	λ_9	-100.03	1.07×10^{-1}	1.07×10^{-1}	8.26×10^{-6}
		σ_4	λ_8	-100.03	1.07×10^{-1}	1.07×10^{-1}	8.26×10^{-6}
		σ_5	λ_1	99.966	0.89	0.89	1.30×10^{-7}
	Σ_2'	σ_6	λ_2	10.013	0.89	0.9	2.36×10^{-7}
		σ_7	λ_7	-9.9848	0.89	0.89	1.97×10^{-7}
Σ_2''	Σ_7	σ_8	λ_3	1.0014	1	1	4.6×10^{-7}
	Σ_8	σ_9	λ_6	-1.0002	1.75×10^{-4}	1.75×10^{-4}	2.84×10^{-4}
	Σ_9	σ_{10}	λ_5	-1.0000	1.75×10^{-4}	1.75×10^{-4}	1.95×10^{-4}
	Σ_{10}	σ_{11}	λ_4	-0.9997	3.91×10^{-4}	3.91×10^{-4}	3.22×10^{-4}

De esta manera el Algoritmo 1 (SSVD) obtiene la descomposición espectral de la matriz A con alta precisión relativa, queda resuelta la problemática que tenía la rutina original para el cálculo de algunos vectores propios al estar el error de estos gobernados por los gaps relativos de valores singulares.

4.3. Resultados de errores para los vectores propios definitivos

Finalmente presentamos el resultado de los errores de los vectores propios para la rutina `SYSVD` definitiva [5]:

Teorema 4.1 *Sea una matriz simétrica $A \in \mathbb{R}^{n\times n}$ de rango r para la cual es posible computar la descomposición SVD con pequeños errores multiplicativos de orden $O(\kappa\,\epsilon)$ satisfaciendo (56) y (57). Sea $\widehat{\Lambda}_j^+$ (resp. $\widehat{\Lambda}_j^-$) una agrupación de valores propios distinta de cero computada en los* `pasos 1` *y* `2` *del algoritmo* `SYSVD` *(con* `tol` $= O(\kappa\epsilon)$, `tolgap` $= 1/2$, `tolgap2` $= 1/(2\,n)$*), sean $\Lambda_j^\pm$ los valores propios exactos correspondientes a la agrupación y $\widehat{Q}_j^+$ (resp. $\widehat{Q}_j^-$) los vectores propios computados en los* `pasos 4` *y* `5` *del algoritmo* `SYSVD`*. Existe una matriz Q_j^+ (resp. Q_j^-) cuyas columnas forman una base ortonormal de subespacio invariante que corresponde con los valores propios en Λ_j^+ (resp. Λ_j^-) de A de modo que se cumple que:*

$$\|\,\widehat{Q}_j^+ - Q_j^+\|_F \;\leq\; \frac{\kappa\,O(n\,\epsilon)}{relgap(\widehat{\Lambda}_j^+)} \tag{85}$$

$$\|\,\widehat{Q}_j^- - Q_j^-\|_F \;\leq\; \frac{\kappa\,O(n\,\epsilon)}{relgap(\widehat{\Lambda}_j^-)}. \tag{86}$$

Asimismo, sea $\widehat{Q} = [\widehat{Q}_1^+\ \widehat{Q}_1^-\ \ldots \widehat{Q}_q^+\ \widehat{Q}_q^-]$ la matriz $n\times r$ cuyas columnas son las bases de todos los subespacios invariantes computados, entonces existe una matriz $\bar{Q}$ con columnas exactamente

ortonormales que cumple:

$$\widehat{Q} = \bar{Q} + H \quad with \quad \|H\|_F = O(\epsilon). \tag{87}$$

Como conclusión, se ha conseguido resolver el problema que presentaba la rutina original, las cotas de errores (71) dependientes de gaps relativos de valores singulares. El algoritmo SSVD obtiene la descomposición espectral con alta precisión relativa (APR), tanto para valores como para vectores propios, siempre que la SVD del primer paso esté calculada con alta precisión relativa.

5. Experimentos numéricos

En esta sección se presentan los resultados de los tres tipos de experimentos numéricos que se han realizado.

- En el primero, se pone a prueba la rutina `SYSVD`, el segundo paso del Algoritmo 1 (SSVD), teniendo controlados los errores en la descomposición SVD.
- En el segundo se pone a prueba el Algoritmo 1 (SSVD) completo, incluyendo la computación de la descomposición SVD.
- Los dos tipos de experimentos también se realizan usando, en el segundo paso del Algoritmo 1 (SSVD), la rutina `SYSVD` en su versión definitiva (Algoritmo 3) y en la primera versión `SYSVD0` (Algoritmo 2). En este último caso nos referiremos al algoritmo como SSVD0. Así se podrá comprobar que los refinamientos mejoran la precisión en el cálculo de los vectores propios para determinadas distribuciones. No obstante, a tal efecto, se ha diseñado el tercer experimento, donde se han generado matrices cuyas distribuciones de valores singulares reproducen lo mostrado en la sección §4.2.2. Para estas distribuciones la mejora respecto a la versión original del algoritmo será muy evidente.

Para describir el modo en que se han desarrollado los experimentos es necesario mencionar una factorización que tiene un papel muy importante en el algoritmo SSVD, ésta no ha sido tratada anteriormente para no sobrecargar este documento cuyo objetivo es introducir a la comunidad no especializada en el Álgebra Lineal Numérica el algoritmo SSVD.

Se trata de la factorización RRD (Rank Revealing Decomposition). Es una factorización que como su propio nombre indica, revela el rango. La factorización RRD de una matiz $G \in \mathbb{R}^{m\times n}$ es:

$$G = XDY^T \tag{88}$$

Siendo $D \in \mathbb{R}^{r\times r}$ diagonal no singular y $X \in \mathbb{R}^{m\times r}, Y \in \mathbb{R}^{n\times r}$ son matrices de rango completo por columnsas bien condicionadas. El rango de la matriz G es precisamente r.

Su relación con este trabajo viene de [4]. Allí se demostró que la condición necesaria y suficiente que debe cumplir una matriz para que pueda obtenerse su descomposición SVD con alta precisión relativa es que se pueda obtener su descomposición RRD con errores regresivos multiplicativos de $O(\epsilon)$. Por ello, debido a que el Algoritmo 1 (SSVD) toma como punto de partida la descomposición SVD, la factorización RRD es esencial para el algoritmo SSVD.

5.1. Implementación del algoritmo SSVD

1. Antes del `paso` 1 del Algoritmo 1 (SSVD), se ha realizado una factorización RRD de la matriz A, $A = XDX^T$, ha sido realizada utilizando una modificación de la descomposición simétrica indefinida de Bunch y Parlett (BP) [1].
2. Para el `paso` 1 del Algoritmo 1 (SSVD), la descomposición SVD, se ha utilizado el Algoritmo 3.1 de `DGESVD` [3], que parte de una RRD. Se han utilizado algoritmos pertenecientes a las librerías **LAPACK** and **BLAS** exclusivamente, incluyendo el código para la one-sided Jacobi code, para la cual se ha usado la rutina desarrollada por Z. Drmač `SSYEVJ`. La

implementación del procedimiento (llamado `SGEPSV` en simple precisión) ha seguido los siguientes pasos:

Algorithm 6 (`SGEPSV`)
`INPUT:` X, D, Y : $A = XDY^T$.
`OUTPUT:` U, Σ, V : $A = U\Sigma V^T$.

`a)` *factorización QR pivotando las columnas de* XD,
$XDP = QR$; $A = QRP^TY^T$
`LAPACK` *Rutina:* `SGEQPF`

`b)` *Obtener* $W = R(YP)^T$; $A = QW$
`BLAS` *Rutina:* `STRMM`

`c)` *SVD de W utilizando one-sided Jacobi;* $W = \overline{U}\Sigma V^T$; $A = Q\overline{U}\Sigma V^T$
`LAPACK` *Rutina:* `SSYEVJ`

`d)` `Obtener` $U = Q\overline{U}$; $A = U\Sigma V^T$
`LAPACK` *Rutina:* `SORMQR`

Con la actual implementación la utilización del algoritmo SVD realizando los dos pasos reflejados en el Algoritmo 1 (SSVD), más el paso inicial extra de la RRD, el tiempo de computación es comparable con el algoritmo J-ortogonal del que se habló previamente.

3. Los algoritmos `SYSVD0` y `SYSVD`, `paso` 2 del Algoritmo 1 (SSVD), ha sido implementados tal y como se ha descrito en la secciones §3 y §4. A continuación se describen algunos detalles específicos de la implementación:

   a) Se recuerda que el cálculo de los valores propios para las rutinas `SYSVD0` y `SYSVD`, la versiones original y mejorada del algoritmo SVD, es exactamente igual, la única diferencia entre ambos reside en el cálculo de los vectores propios.

   b) La elección de las agrupaciones inicial se ha realizado tomando $C = 1$ en la expresión (60) siendo κ una constante relacionada con números de condición de las matrices involucradas en el algoritmo `SGEPSV` [6]

   $$\kappa \lesssim \kappa(X)\kappa(R). \tag{89}$$

   Para su cálculo se ha utilizando el estimador $O(n^2)$ de la rutina de **LAPACK** `STRCON` para estimar $\kappa(R)$ y $\kappa(X)$ cuando se ha partido de una matriz no factorizada. En ocasiones se han generado matrices en la forma con $\epsilon\kappa$ mayor que 1. Lo que quiere decir que la rutina que calcula la SVD, Algoritmo 6, no garantiza ninguna cifra significativa al computar los valores singulares. De hecho, la expresión (29) en este caso nos dice que todos los valores singulares estarán contenidos en un agrupación. Por tanto el criterio seguido para incluir dos valores singulares contiguos σ_j, σ_{j+1} en el mismo agrupación ha sido:

$$\frac{|\sigma_j - \sigma_{j+1}|}{\sigma_j} \leq \text{mín}\{\epsilon\kappa, 1/n\}. \tag{90}$$

c) El producto $\Delta_i = V_i^T U_i$ realizado en el `paso` 11 del Algoritmo 2.3 se ha realizado utilizando la rutina de BLAS `SGEMM`.

d) La diagonalización de las submatrices $\Delta_i = [W_i^+ \, W_i^-] J_i [W_i^+ \, W_i^-]^T$ se ha realizado mediante la rutina `SSYEV` de **LAPACK** aplicada exclusivamente a la mitad triangular superior de la matriz Δ_i. Finalmente, el cálculo de los vectores propios a través del producto $Q_i^{\pm} = V_i W_i^{\pm}$ se ha obtenido usando la rutina `SGEMM` de **BLAS**.

e) En todos los experimentos el valor para el parámetro `tolgap` que aparece que se utiliza en la segunda selección de agrupaciones se ha fijado como `tolgap = 1/2` y el parámetro `tolgap2` para la tercera selección de agrupaciones se ha fijado como `tolgap2 = 1/(2(n-1))`.

5.2. Resultados numéricos

Los siguientes experimentos numéricos se ha realizado utilizando una procesador Intel Core 2 T5300 con aritmética IEEE, y las rutinas fueron implementadas utilizando Visual Studio 2005 de Microsoft. Todos los experimentos numéricos en esta sección han sido realizados para matrices no singulares aunque como se dijo en la sección §3.2.3 el algoritmo SVD puede ser aplicado a matrices con deficiencia de rango.

Para probar el Algoritmo 1 (SSVD) se ha tomado como referencia los valores y vectores propios obtenidos con la rutina desarrollada por I. Slapničar y K. Veselić [13, 11], que implementa el algoritmo one-sided Jacobi implícito en doble precisión. En lo sucesivo nos referiremos a estos valores y vectores propios simplemente como λ_i y q_i, que para efectos prácticos podemos considerar exactos. Estos vectores serán comparados con los valores y vectores propios $\lambda_i^{(\mathtt{M})}$ y $q_i^{(\mathtt{M})}$ computados en simple precisión con las siguientes rutinas[8]:

- Algoritmo 1 (SSVD), usando como paso las rutinas original y mejorada, $\lambda_i^{(\mathtt{SO})}$ y $q_i^{(\mathtt{SO})}$ obtenidos con `SYSVD0` y $\lambda_i^{(\mathtt{S})}$ y $q_i^{(\mathtt{SO})}$ obtenidos con `SYSVD` respectivamente. Recordamos aquí nuevamente que las rutinas estas dos rutinas solo difieren el el procedimiento por el que se calculan los vectores propios, los valores propios se obtienen de la misma manera, $\lambda_i^{(\mathtt{SO})} = \lambda_i^{(\mathtt{S})}$.
- Rutina `SSYEVJ`, el algoritmo J-Ortogonal, obteniendo $\lambda_i^{(\mathtt{J})}$ y $q_i^{(\mathtt{J})}$.
- La rutina `SSYEV` de **LAPACK**, que implementa el algoritmo QR, $\lambda_i^{(\mathtt{QR})}$ y $q_i^{(\mathtt{QR})}$.

Para estos métodos se han medido las siguientes cantidades, para cada matriz generada para la que se ha resuelto la descomposición espectral.

1. El máximo error relativo en los valores propios:

$$e_\lambda^{(\mathtt{M})} = \max_i \left| \frac{\lambda_i - \lambda_i^{(\mathtt{M})}}{\lambda_i} \right|. \tag{91}$$

[8] El superíndice `(X)` indica el método con el que se obtuvo la descomposición espectral, `X`:= `J` (J-Ortogonal), `S` (SSVD), `S0` (SSVD0), `QR` (Algoritmo QR, rutina SSYEV de LAPACK).

2. Una cantidad de control para los valores propios:

$$\vartheta^{(\mathtt{M})} = \frac{e_{\lambda}^{(\mathtt{M})}}{\kappa\,\epsilon}, \tag{92}$$

donde κ viene de (89) y $\epsilon = 5.96\,10^{-8}$, es la unidad de redondeo en precisión sencilla. Se espera que $\vartheta^{(\mathrm{S})}$ sea de orden $O(1)$ debido a (66) en el Teorema 3.6.

3. El error en las bases de subespacios invariantes[9]

$$E_{\Lambda_i}^{(\mathtt{M})} = \|\sin\Theta(Q_i, Q_i^{(\mathtt{M})})\|_2. \tag{93}$$

En el caso de que las agrupaciones cuenten con sólo un elemento hemos computado simplemente la norma euclidiana de la diferencia entre los vectores propios computados en doble y simple precisión, q_i y $q_i^{(\mathtt{M})}$, respectivamente:

$$e_{q_i}^{(\mathtt{M})} = \|q_i - q_i^{(\mathtt{M})}\|_2. \tag{94}$$

Realmente, las cantidades $e_{q_i}^{(\mathtt{M})}$ son siempre computadas aún en el caso de que la dimensión de los agrupaciones sea mayor de 1. Se hace esto para comprobar que los agrupaciones se forman sólo cuando no se puede garantizar precisión para los vectores propios calculados individualmente (ver Sección §4).

4. La cantidades de control para las bases de subespacios invariantes son:

$$\Xi_{\Sigma}^{(\mathtt{M})} = \max_i \frac{E_{\Lambda_i}^{(\mathtt{M})}\ relgap(\Sigma_i^{(\mathtt{M})})}{\kappa\,\epsilon}, \tag{95}$$

$$\Xi_{\Lambda}^{(\mathtt{M})} = \max_i \frac{E_{\Lambda_i}^{(\mathtt{M})}\ relgap(\Lambda_i^{(\mathtt{M})})}{\kappa\,\epsilon}, \tag{96}$$

y los correspondientes para los vectores propios individuales:

$$\xi_{\sigma}^{(\mathtt{M})} = \max_i \frac{\|q_i - q_i^{(\mathtt{M})}\|_2\ relgap(\sigma_i^{(\mathtt{M})})}{\kappa\,\epsilon}, \tag{97}$$

$$\xi_{\lambda}^{(\mathtt{M})} = \max_i \frac{\|q_i - q_i^{(\mathtt{M})}\|_2\ relgap(\lambda_i^{(\mathtt{M})})}{\kappa\,\epsilon}. \tag{98}$$

Las cantidades (96) y (98) sirven para comprobar las relaciones (85) y (86) en el Teorema 4.1, mientras que (95) y (97) hacen lo propio para (71) en el Teorema 3.8. Estas últimas sólo se han computado para los métodos SSVD y SSVD0. Para realizar los experimentos se han generado matrices en simple precisión de maneras diferentes. Todas las matrices aleatorias necesarias se han generado utilizando rutinas de **LAPACK** `SLATM1`, para matrices diagonales, y `SLATMR`, para matrices completas. Cuando se han generado matrices con un determinado número de condición $\mathcal{K}$, esto se ha hecho produciendo matrices diagonales con elementos cuyos valores

[9] $\|\sin\Theta(Q_i, Q_i^{(\mathtt{M})})\|_2$ es una distancia entre los espacios columnas de Q_i y $Q_i^{(\mathtt{M})}$ y es llamada el menor ángulo principal (ver [14, Sec 7.5]). Es necesaria usarla cuando tratamos con subespacios invariantes de dimensión mayor que 1.

absolutos dentro del rango 1 a $1/\mathcal{K}$ para posteriormente, multiplicar por una matriz ortogonal aleatoria en simple precisión. La distribución de los elementos de la diagonal esta controlada por el parámetro `MODE` de la rutina `SLATM1`: $|\texttt{MODE}| = 3$, distribución geométrica; $|\texttt{MODE}| = 4$, distribución aritmética; $\texttt{MODE} = 5$, con distribución logarítmica. Si `MODE` es positivo o negativo, los elementos son dispuestos en orden decreciente o creciente, respectivamente.

Se han generado matrices mal condicionadas, esto es, con números de condición elevados, para probar la dependencia de los resultados de los Teoremas 3.6, 3.8 and 4.1 con el parámetro κ.

5.2.1. Experimento 1

Este experimento ha sido diseñado para probar la segunda etapa del Algoritmo 1 (SSVD) ya que para la primera etapa se han controlado los errores generando matrices para obtener una factorización RRD con un determinado condicionamiento. Para la segunda etapa se han utilizado tanto la rutina original Algoritmo 2 `(SYSVD0)` como la rutina definitiva Algoritmo 3 `(SYSVD)`.

Hemos generado las matrices X y D para configurar la factorización RRD de la matriz A, $A = XDX^T$, donde X es una matriz $n \times n$ y D es una matriz diagonal. Los parámetros se han seleccionado del siguiente modo: $\kappa(X) = 10^{[2:1:6]}$; $\kappa(D) = 10^{[2:2:16]}$; $MODE_X = 3, 4, 5$; $MODE_D = \pm 3, \pm 4, 5$. Para cada conjunto de parámetros se han generado 20 matrices para las dimensiones $n = 50, 100$ (12000 matrices para cada n), 2 para $n = 250$ (1200 matrices), 2 para $n = 500$ (1200 matrices), 1 para $n = 1000$, y solamente 2 MODES (80 matrices).

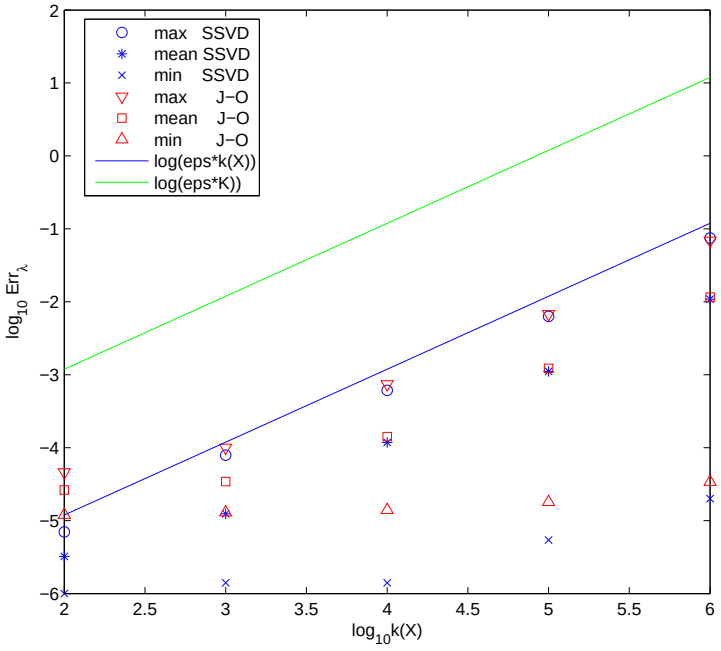

Figura 5: Experimento 1. Máximo error relativo de los valores propios: $\log_{10} e_\lambda$ vs. $\log_{10} \kappa(X)$

La Figura 5 muestra el máximo, mínimo y la media (sobre todos los $MODE$s, muestras, y nº de condición $\kappa(X)$) de la cantidad $log_{10}e_\lambda$, que representa aproximadamente el número de dígitos correctos en la cálculo de los valores propios como función de $\kappa(X)$, el número de condición de la matriz X para $n = 100$. Se muestran los resultados obtenidos para dos algoritmos: el algoritmo 1 (SSVD) y el algoritmo J-ortogonal. La línea $\epsilon\kappa$ se ha dibujado como guía. Los resultados confirman las cotas de errores teóricas para los valores propios (66).

La Tabla 5 muestra la información estadística que corresponde a la cantidad $\vartheta^{(\mathrm{S})}$. El objetivo es estudiar la cota de error (66) para el Algoritmo 1 (SSVD) y comparar su precisión con el algoritmo J-ortogonal. La información mas significativa de la Tabla 5 aparece bajo la columnas con etiqueta "máx" donde se presentan los máximos valores para cada magnitud. En particular, el hecho de la cantidades de la primera columna sean menos que 1 confirma que el algoritmo 3 satisface la cota de error (66).

Se puede apreciar en la Figura 5 y en la Tabla 5, que el Algoritmo 1 (SSVD) obtiene los valores propios igual de bien (o incluso mejor, especialmente para pequeños valores de $\kappa(X)$)) que el algoritmo J-ortogonal , donde el máximo error para el Algoritmo 1 (SSVD) se ajusta muy bien al comportamiento teórico $\epsilon\kappa$. También se puede observar que no hay dependencia con n.

Tabla 5: Exp 1. Datos estadísticos para la precisión de los valores propios: $\vartheta^{(\mathtt{M})}$.

n	50		100		250		500		1000	
Método	media	máx	media	máx	media	máx	media	máx	media	máx
$\vartheta^{(\mathtt{S})}$	.032	.57	.025	.3	.021	.2	.022	.26	.018	.26
$\vartheta^{(\mathtt{J})}$	.095	1.1	.12	1.7	.18	2.5	0.24	3.5	.27	4

Para una fracción significativa de todas las matrices (4143 matrices de 12000 para $n = 50$; 6727 matrices de 12000 para $n = 100$; 981 matrices de 1200 para $n = 250$; 1097 matrices de 1200 para $n = 500$; 78 matrices de 80 para $n = 1000$), se han encontrado agrupaciones de valores singulares de dimensión mayor a 1, así mismo la agrupación de mayor dimensión que se ha encontrado es 5. La media de las agrupaciones de dimensión mayor a 1 está en el rango de casi ninguna para $n = 50$ y aproximadamente 40 para $n = 1000$, con una dimensión típica de 2. Esto demuestra que el criterio que se utiliza la primera selección de agrupaciones (2.1) (la única que afecta al cálculo de los valores singulares) es correcto ya que se determinan en la práctica los signos correspondientes a todos los valores propios. Después de aplicar la segunda y tercera selección de agrupaciones todas las matrices consideradas presentan agrupaciones, la media de agrupaciones que se presentan en una matriz es $0.3n$ para todos los n.

Ahora centramos en análisis en lo concerniente a los vectores propios y a los subespacios invariantes. La Tabla 6 presenta las cantidades $\Xi_\Sigma^{(\mathtt{M})}$ y $\Xi_\Lambda^{(\mathtt{M})}$ definidas anteriormente para el algoritmo SYSVD en sus dos versiones, los Algoritmos 2 (`SYSVD0`) y 3 (`SYSVD`). Para el algoritmo J-ortogonal se presenta la cantidad que gobierna su error: $\Xi_\Lambda^{(\mathrm{S})}$. Cuando se comparan los resultados de las rutinas `SYSVD0` y `SYSVD` con sus respectivos relgaps de valores singulares (fila 1), se puede ver que ambos métodos se comportan de la manera esperada. Cuando se compara la rutina `SSYSVD0` con los gaps relativos de valores propios los resultados pueden no ser buenos

(ver fila 2).[10] Cuando se usa el algoritmo `SYSVD` los resultados mejor significativamente (fila 4 respecto a fila 2), mostrando que el método computa las bases para estas matrices con errores gobernados por los gaps relativos de valores propios, del mismo modo que lo hace el algoritmo J-ortogonal.

Tabla 6: Exp 2. Datos estadísticos para la precisión de los valores propios: $\vartheta^{(\mathtt{M})}$.

n	50		100		250		500		1000	
Método	media	máx	media	máx	media	máx	media	máx	media	máx
$\Xi_{\Sigma}^{(\mathtt{SO})}$	.032	.46	.05	1.1	.085	1.8	.12	5.1	.084	1.2
$\Xi_{\Lambda}^{(\mathtt{SO})}$	.39	230	0.84	660	2.4	6700	6.662	1100	9.5	330
$\Xi_{\Sigma}^{(\mathtt{S})}$	.035	.46	.055	1.3	.093	2.5	.13	5.1	.089	1.4
$\Xi_{\Lambda}^{(\mathtt{S})}$	.042	.6	.074	3.7	.15	4.8	.23	6.9	.24	6.7
$\Xi_{\Lambda}^{(\mathtt{J})}$	.045	.7	.077	1.3	.15	3	.22	5.5	.18	3.2

La Tabla 7 muestra las cantidades $\xi_{\sigma}^{(\mathtt{M})}$ y $\xi_{\lambda}^{(\mathtt{M})}$ definidas en (97) y (98). Estas son cantidades que se refieren a las errores de los vectores propios vector por vector. Se puede observar que la precisión en los vectores propios no se ve deteriorada por los procesos de selección de agrupaciones implícitos en los Algoritmos 2 (`SYSVD`) y 3 (`SYSVD0`). Comentarios similares a los realizados para la Tabla 6 se pueden aplicar igualmente aquí.

Tabla 7: Exp 1. Datos estadísticos para la precisión de los vectores propios: $\xi_{\sigma}^{(\mathtt{M})}$ and $\xi_{\lambda}^{(\mathtt{S})}$.

n	50		100		250		500		1000	
Método	media	máx	media	máx	media	máx	media	máx	media	máx
$\xi_{\sigma}^{(\mathtt{SO})}$	.034	.76	.056	1.3	.091	1.8	.13	5.1	.084	1.2
$\xi_{\lambda}^{(\mathtt{SO})}$	.39	230	0.854	660	2.5	670	6.7	1100	9.5	330
$\xi_{\sigma}^{(\mathtt{S})}$	.035	1.5	.063	1.5	.10	1.8	.15	5.1	.094	1.2
$\xi_{\lambda}^{(\mathtt{S})}$	.046	1.5	.089	3.7	.17	4.8	.27	6.9	.25	6.7
$\xi_{\lambda}^{(\mathtt{J})}$	.045	.7	.077	1.3	.15	3	.22	5.5	.18	3.2

5.2.2. Experimento 2

En este experimento el objetivo es probar el Algoritmo 1 (SSVD) completo, ya que al contrario del Experimento 1, aquí se parte de la matriz completa sin factorizar y el control de los errores de la primera etapa es menor. Se sabe que sobre matrices de la forma $A = DBD$ (se las conoce con el nombre de graduadas o escaladas), donde D es diagonal y B es simétrica, muchas veces se puede calcular la SVD con alta precisión relativa siempre que B esté bien condicionada aunque A esté muy mal condicionada.

Para este segundo experimento se han generado matrices simétricas multiplicando matrices aleatorias bien condicionadas, B, por matrices diagonales aleatorias mal condicionadas, D. No

[10] De cualquier como, como se puede deducir de la media de $\Xi_{\Lambda}^{(\mathtt{SYSVD0})}$, matrices para las cuales `SYSVD0` calcula los vectores propios con errores grandes respecto a se relgaps de valores propios son muy infrecuentes

siempre es posible obtener una factorización SVD con alta precisión relativa para este tipo de matrices, pero en la práctica se consiguen frecuentemente errores en los valores singulares de orden $O(\epsilon\kappa(B))$. Por lo que para las matrices generadas en este experimento se espera obtener de los valores y vectores con alta precisión relativa.

Los parámetros que se han elegido son los siguientes: $\kappa(B) = 10^{[0:1:3]}$, $\kappa(D) = 10^{[2:2:10]}$, $MODE_B = 3, 4, 5$, $MODE_D = \pm 3, \pm 4, 5$. Para cada juego de parámetros hemos probado 50 matrices para $n = 50, 100$ (un total de 15000 matrices para cada n), 5 para $n = 250, 500$ (un total de 1500 matrices para cada n), 1 para $n = 1000$, y únicamente 5 combinaciones de los $MODE$s (un total de 100 matrices). En este experimento también se presentan los resultados obtenidos con el algoritmo QR para comparar nuestro algoritmo con un algoritmo que garantizan precisión absoluta.

Se presentan las mismas magnitudes que para el experimento anterior, las Tablas 8 y 9, presentan los errores en el cálculo de los valores y de los vectores propios. Los resultados para las bases de los subespacios invariantes no se presentan pues son prácticamente iguales que que los obtenidos para los vectores propios.

Nótese que los máximos valores en la Tabla 8 son mayores que para la tabla análoga en el primer experimento tanto para el algoritmo SSVD como para el J-ortogonal. Esto se debe a la inclusión del error la factorización inicial que se comporta aproximadamente como $O(\epsilon\kappa(B))$. En cualquier caso, ambos algoritmos, se comportan mucho mejor que el algoritmo QR, para este tipo de matrices. Un detalle interesante es que las cantidades $\vartheta^{(\mathrm{S})}$ en la Tabla 8 disminuyen a medida que n aumenta. Esto ocurre porque en este experimento el número de condición κ decrece con la dimensión n mas rápido que los errores relativos de valores propios $e_\lambda^{(\mathtt{S})}$. Las cantidades de control de los vectores propios mostradas en las Tablas 9 también decrecen con n por la misma razón. De todos modos, los máximos valores de las cantidades de control para estos valores propios (Tabla 8) son mucho mayores que para los vectores propios (Tabla 8), eso fenómeno no es explicado por las cotas de errores.

Tabla 8: Exp 2. Datos estadísticos para la precisión en los valores propios:$\vartheta^{(\mathrm{S})}$.

n	50		100		250	
Método	media	máx	media	máx	media	máx
$\vartheta^{(\mathtt{S})}$	2.21	4000	1.22	3600	0.27	120
$\vartheta^{(\mathtt{J})}$	2.29	500	.1.49	7000	0.31	100
$\vartheta^{(\mathtt{QR})}$	$1.^{13}$	$2 \cdot 10^{16}$	$7 \cdot 10^{11}$	$1 \cdot 10^{14}$	$1 \cdot 10^{10}$	$8 \cdot 10^{12}$

n	500		1000	
Método	media	máx	media	máx
$\vartheta^{(\mathtt{S})}$	0.12	5	.038	1.7
$\vartheta^{(\mathtt{J})}$	0.13	5, 8	.068	3
$\vartheta^{(\mathtt{QR})}$	$8 \cdot 10^{12}$	$5 \cdot 10^{12}$	$3 \cdot 10^3$	$2 \cdot 10^5$

Tabla 9: Exp 2. Datos estadísticos para la precisión en los vectores propios: $\xi_\sigma^{(\mathrm{S})}$ y $\xi_\lambda^{(\mathrm{S})}$.

n	50		100		250		500		1000	
Método	media	máx	media	máx	media	máx	media	máx	media	máx
$\xi_\sigma^{(\mathtt{SO})}$	.47	11	.28	4.6	.17	1.1	.064	.55	.023	.16
$\xi_\lambda^{(\mathtt{SO})}$	3.6	3300	2.8	1900	1.2	1600	.30	14	.067	.51
$\xi_\sigma^{(\mathtt{S})}$	.47	11	.31	5.2	.20	1.1	.076	1.3	.024	.16
$\xi_\lambda^{(\mathtt{S})}$	.56	12	.34	5.8	.25	2.4	.091	1.3	.030	.16
$\xi_\lambda^{(\mathtt{J})}$	.60	21	.37	4.3	.17	1.2	.090	.67	.039	.20

Del mismo modo que ocurre en el Experimento 1, para una gran cantidad de matrices (323 matrices de 15000 para $n = 50$; 4783 matrices de 15000 para $n = 100$; 1023 matrices de 1500 para $n = 250$; 1471 matrices de 1500 para $n = 500$; 100 matrices de 100 para $n = 1000$), se encuentran agrupaciones de valores singulares con dimensión mayor a 1, con una dimensión máxima de 5. La media de agrupaciones con dimensión mayor que 1 está en el rango de casi ninguna para $n = 50$ a aproximadamente 60 agrupaciones para $n = 1000$ con una dimensión típica de 2. Esto demuestra nuevamente que el criterio que se utiliza en la primera selección de agrupaciones (2.1) (la única que afecta al cálculo de los valores propios) es correcto ya que determinan correctamente los signos correspondientes a todos los valores propios. Después de aplicar la segunda y tercera selección de agrupaciones todas las matrices consideradas presentan agrupaciones, la media de agrupaciones que se presentan en una matriz es 0.3 n para todos los n.

5.2.3. Experimento 3

Los dos experimentos anteriores han servido para ilustrar que la rutina, al proporcionar alta precisión relativa, es capaz de obtener la descomposición espectral de una matriz mal condicionada de forma precisa. En este experimento el objetivo es probar las mejoras que se han realizado para solucionar las limitaciones de la rutina SYSVD original. Según el planteamiento original de la rutina, tal y como hemos visto en la sección §4, se podían obtener vectores propios poco precisos por razones que nada tienen que condicionamiento de las matrices, sino con procesos internos del algoritmo.

Brevemente, debido a que el algoritmo SSVD toma como punto partida la descomposición SVD, las cotas de errores de los los vectores singulares están asociadas a sus gaps relativos de valores singulares, para determinadas distribuciones de valores singulares y valores propios, estos gaps relativos de valores singulares pueden ser mucho menores que los correspondientes de gaps relativos de valores propios concurriendo de esta manera en una pérdida de precisión en la obtención de determinados vectores propios.

Puesto que estos problemas de precisión en el cálculo de los vectores propios no responden a un mal condicionamiento de matrices, los algoritmos convencionales que no proporcionan APR (QR, Jacobi...) no presentan ningún problema para estas situaciones.

Por tanto en este experimento no se han generado matrices con un determinado condicionamiento sino con una determinada distribución de valores singulares y valores propios. Estas distribuciones se han generado para reproducir situaciones en las que la rutina original SYSVD0

produce vectores propios poco precisos (75) y en las que es necesario realizar los procesos descritos en la sección §4 para obtener unos vectores adecuados: la creación de macroagrupaciones y la obtención de una nueva colección de vectores sobre los que proyectaremos los vectores poco precisos.

Para este experimento hemos considerado los algoritmos SSVD0 y SSVD para ilustrar las mejoras, así como el algoritmo J-Ortogonal que se utilizará como referencia, ya que, al igual que los algoritmos convencionales, no presenta problemas para estas particulares distribuciones de valores singulares y valore propios.

El mínimo error con el que un algoritmo puede obtener los vectores propios es la unidad de redondeo, que en simple precisión toma le valor de $\epsilon = 5.89 \cdot 10^{-8}$. Para un algoritmo que proporciona APR (74) la precisión que se pierde se debe a la inevitable contribución del gap relativo de valores propios y a procesos internos del algoritmo que se representan como una constante κ que es de esperar que sean de orden $O(1)$.

La Figura 6 muestra, para 150 matrices generadas de forma que se presenten macroagrupaciones (tal y como las hemos definido en la sección §4) el mínimo error en el cálculo de los vectores propios obtenidos con los algoritmos SSVD0, SSVD y J-Ortogonal en ordenadas y diferentes casos en abscisas.

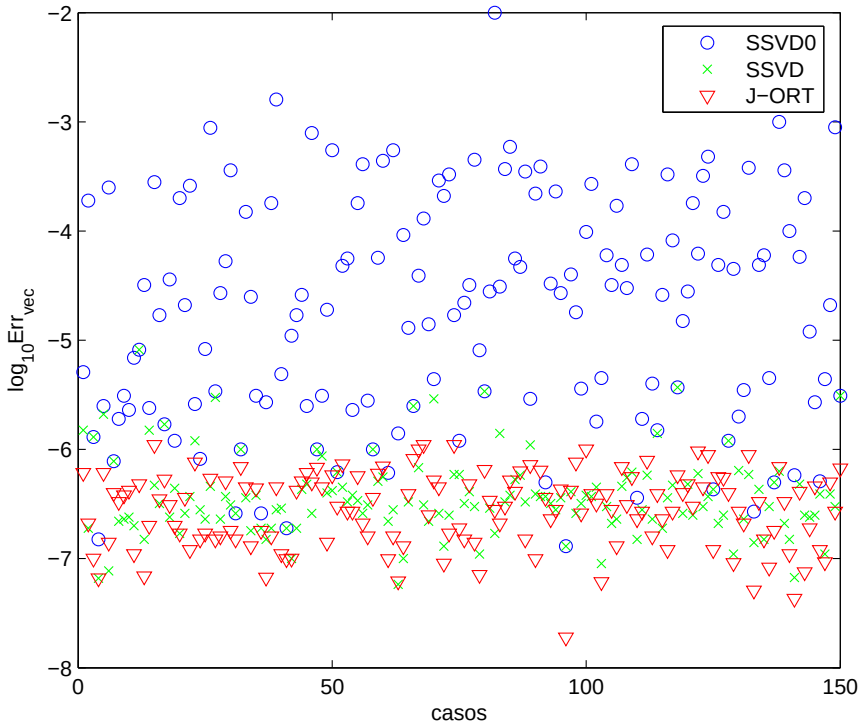

Figura 6: Tercera selección de agrupaciones de valores singulares

Puede observarse como el algoritmo SSVD definitivo (triángulos rojos) recupera la precisión que pierde el algoritmo SSVD original (circunferencias azules), que pueden llegar a ser hasta cuatro dígitos, hasta obtener la precisión completa que tiene el algoritmo J-Ortogonal (cruces verdes). Estos resultados ratifican la cotas de error que se presentaban en (85) y (86), donde a costa de un una constante cuyo orden de magnitud es a lo sumo de orden $O(n)$ (habitualmente mucho menor) se resuelve el problema en la precisión de los vectores propios derivado de partir de la descomposición SVD

Se han generado matrices de dimensión n en un intervalo $n \in [4, 35]$. La agrupación más grande ha sido de 11 valores singulares y las matrices han presentado entre 1 y 3 macroagrupaciones con un valor típico de 1.

Conviene comentar que si bien las matrices mal condicionas se pueden presentar a menudo, por ejemplo en la resolución un problema relacionado con un fenómeno físico como es el calculo de las frecuencias de vibración (valores propios) del ala de un avión, encontrarse con estas particulares distribuciones de valores singulares y valores propios como se han generado es estas matrices es improbable, pero no imposible. En cualquier caso el importante resultado aquí presentado es que es posible resolver esa limitación.

6. Conclusiones

Resumamos este recorrido por el algoritmo SSVD con las ideas más importantes expuestas en este trabajo:

- El algoritmo SSVD, al igual que el algoritmo de Jacobi implícito [7], calcula la descomposición espectral para una matriz simétrica general con Alta Precisión Relativa para todas para las que se pueda calcular una RRD con alta precisión. De momento esta es la clase de matrices más grandes para las que esto se puede hacer.
- El algoritmo SSVD toma como punto de partida otro importante problema del Álgebra Lineal, la Descomposición SVD. Para la Descomposición SVD se han llegado a importantes resultados en lo que a Alta Precisión Relativa se refiere, tanto en identificación de matrices con las que pude obtenerse la descomposición con APR como los algoritmos que lo consiguen. La idea principal del algoritmo SSVD es aprovecharse de esos resultados utilizando las analogías que plantean en el caso simétrico ambas descomposiciones.
- Si bien el planteamiento por el cual se obtiene la Descompresión Espectral a partir de Descomposición SVD (rutina SYSVD) es para la gran mayoría de las matrices simétricas muy sencillo, para determinadas distribuciones de valores singulares y propios es posible encontrar problemas para obtener vectores propios. A través de unos innovadores planteamientos desarrollados por los autores del algoritmo SSVD a lo largo de su estudio [5], se ha conseguido resolver esta limitación que inicialmente se planteaba.

El trabajo que en este documento se presenta es completo, pero no es un trabajo cerrado. Su proyección futura inmediata es constituir un punto de partida sólido para presentar formalmente el algoritmo SSVD definitivo a la comunidad matemática especializada:

- Mediante un nuevo artículo, que se publicará próximamente [5], presentando el nuevo algoritmo como continuación del artículo de 2003.
- La posible inclusión del algoritmo SSVD en la próxima edición de la librería LAPACK, la librería de rutinas de álgebra lineal de libre acceso más importante. El presente trabajo representa el primer paso para alcanzar este objetivo.

Finalmente, comentar que a través del estudio del algoritmo SSVD, se han planteado nuevas líneas de investigación dentro del ALN que podrán ser objeto de estudio durante los próximos años. Algunas de estas líneas de investigación están relacionadas con abordar parcelas del problema de valores propios no simétrico con APR (matrices antisimétricas, matrices normales..) para el que actualmente no se han obtenido resultados concluyentes.

7. Bibliografía

Referencias

[1] J. R. Bunch and B. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal., 8 (1971), pp. 639–655.

[2] J. Demmel, *Accurate singular value decompositions of structured matrices*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 562–580.

[3] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač, *Computing the singular value decomposition with high relative accuracy*, Linear Algebra Appl., 299 (1999), pp. 21–80.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[5] F. M. Dopico and J. M. Molera, *Eigenvectors with full high relative accuracy in the SSVD algorithm*, To be published.

[6] F. M. Dopico, J. M. Molera, and J. Moro, *An orthogonal high relative accuracy algorithm for the symmetric eigenproblem*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 301–351.

[7] P. Dopico, F. M. Koev and J. M. Molera, *Implicit Standard Jacobi Gives High Relative Accuracy*, Accepted in Numerische Mathematik, to appear.

[8] Z. Drmač and K. Veselić, *New fast and accurate Jacobi SVD algorithm: I and II.* LAPACK Working Notes 169 and 170, http://www.netlib.org/lapack/lawns169, http://www.netlib.org/lapack/lawns170.

[9] S. Eisenstat and I. Ipsen, *Relative perturbation techniques for singular value problems*, SIAM J. Numer. Anal., 32 (1995), pp. 1972–1988.

[10] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 3rd ed., 1996.

[11] I. Slapničar, *Accurate symmetric eigenreduction by a Jacobi method*, PhD thesis, Fernuniversität - Hagen, Hagen, Germany, 1992.

[12] L. Trefethen and D. Bau, *Numerical linear algebra*. SIAM, 1997.

[13] K. Veselić, *A Jacobi eigenreduction algorithm for definite matrix pairs*, Num. Math., 64 (1993), pp. 241–269.

[14] D. Watkins, *Fundamentals of Matrix Computations*, Wiley, 1991.

8. Anexo: Código FORTRAN de la rutina SYSVD

```
1         SUBROUTINE SSYSVD(JOBVECTOR, REFINE,ORDER,LDA, M, N, TOL, TOLGAP,
2        %U, S, V, INCLUST, ILCLUST, INCLUST2,MAXDIMCLUST, RGAPS, RGAPSMIN,
3        %WRK1, WRK2, WRK3, IKEY,INFO,in2)
4  *
5  *  -- SYSVD driver routine (version 2.0) --
6  *     Germán Villanueva Baschwitz y Juan Manuel Molera
7  *     Univ. Carlos III de Madrid,
8  *     Department of Mathematics
9  *     June 2009
10 *
11 *     .. Scalar Arguments ..
12       CHARACTER*1       REFINE, ORDER, JOBVECTOR
13       INTEGER           LDA, M, N, INFO, MAXDIMCLUST
14       REAL              TOL,TOLGAP,RGAPSMIN
15 *     ..
16 *     .. Array Arguments ..
17       INTEGER           INCLUST(LDA,*),ILCLUST(*), IKEY(*)
18       REAL              U(LDA,*), S(*), V(LDA,*), RGAPS(LDA,*)
19       REAL              WRK1(*),WRK2(LDA,*),WRK3(LDA,*)
20       INTEGER           INCLUST2(LDA,*)
21 *     ..
22 *
23 *  Purpose
24 *  =======
25 *
26 *  SSYSVD computes all eigenvalues and, optionally, eigenvectors of a
27 *  real symmetric matrix A, given its SVD decomposition.
28 *
29 *                A =  U * diag(S) * V^T
30 *
31 *  It is the second step of the SSVD algortihm
32 *
33 *   Input: Symmetric matrix A.
34 *   Output: Eigenvalues  and eigenvectors of A.
35 *
36 *  1.  Compute SVD of A.
37 *  2.  Compute the eigenvalues and eigenvectors of A from singular values and
38 *         singular vectors using routine SYSVD
39
40 *  This is the method described in the papers:
41 *  [1] "An Orthogonal High Relative Accuracy Algorithm for the
42 *      Symmetric Eigenproblem"  F. M. Dopico, J. M. Molera and J. Moro
43 *      SIAM Journal on Matrix Analysis and Applications,
44 *      Vol 25,n 2 pp 301-351. All the details are in this paper.
45 *  [2] "Eigenvectors with full high relative accuracy in the SSVD algorithm"
46 *       F. M. Dopico and  J. M. Molera.
47 *      In preparation.
48 *
49 *  Here follows a brief description of the  main steps in the routine SYSVD
50 *
51 *  ************************************************************************
52 *  SYSVD pseudocode
```

```
53  *
54  *  Input: Singular value decomposition of a symmetric matrix A.
55  *  Output: Eigenvalues  and eigenvectors of A.
56  *
57  *  1. Decide the singular value clusters (First selection). Tolerance TOL.
58  *     TOL is related to the relative accuracy of the computed singular
59  *     values. If the relative separation between two consecutive
60  *     singular values is less than TOL they have to be
61  *     considered equal: they belong to the same cluster
62  *
63  *  2. Compute the eigenvalues.
64  *     Once the sing. values, and the clusters to which they belong,
65  *     are known, signs are assigned  to them. The signs come from
66  *     diagonalizing the orthogonal symmetric block diagonal matrix
67  *     V^T*U (so all these blocks have eigenvalues  +1 or -1).
68  *     The size of the blocks is determined by
69  *     the number of sing. values in a cluster.
70  *
71  *  3. Merge, when necessary, some pairs of clusters (Second selection).
72  *     Tolerance TOLGAP.
73  *     This first refining of the cluster selection (second refining is step 5)
74  *     is made to improve the relgap of the new cluster in such a way that
75  *     the computed e-vectors (in step 4) have better accuracy.
76  *     If one cluster fullfils
77  *          a. It has eigenvalues of only one sign
78  *          b. relgap(Sigma)/relgap(Lambda) < TOLGAP
79  *          c. Its relative closest, previous or next, cluster has only
80  *             eigenvalues of the oposite sign.
81  *          d. The cluster of the union of the two clusters have relgap(Sigma)
82  *             bigger than the minimum of both
83  *    both clusters are joined in only one.
84  *
85  *  4. Compute the eigenvectors.
86  *     The eigenvectors are obtained from multiplying the singular vector
87  *     matrix by the eigenvectors of the matrices V^TU of step 2.
88  *
89  *  5. Make a new selection of clusters (Third selection) and get
90  *     new eigenvectors. If for an eigenvalue cluster "Lambda"
91  *     inside a singular value cluster "Sigma",
92  *     relgap(Sigma)/relgap(Lambda)< 1/(2*(n-1))
93  *     a new "macrocluster" can be formed around it such that
94  *     the accuracy  for the eigenvectors is as best as possible
95  *     (controlled by 1/relgap(Lambda)).
96  *  ***********************************************************************
97  *
98  *
99  *
100 *  ***********************************************************************
101 *
102 *
103 * Arguments
104 * =========
105 *  JOBVECTOR  (input) CHARACTER*1
106 *             = 'N':  Compute eigenvalues only;
```

```
107 *              = 'Y':  Compute eigenvalues and eigenvectors.
108 *
109 *  REFINE      (input) CHARACTER*1
110 *              = 'Y':  Apply refining steps 3 and 5
111 *              = 'N':  It does not apply the refining method, this is
112 *                      the only possibility if JOBVECTOR='N'.
113 *
114 *  ORDER       (input) CHARACTER*1
115 *              = 'Y': The singular values of A in array S are ordered
116 *                     decreasingly on input.
117 *              = 'N': On input the singular values are not ordered.
118 *
119 *  LDA         (input) INTEGER
120 *              The leading dimension of the two dimensional arrays.
121 *                LDA >= max(1,M).
122 *
123 *  M           (input) INTEGER
124 *               The order of the matrix A.  M >= N >= 0.
125 *
126 *  N           (input/output) INTEGER
127 *               on entry: The number of columns of S, V. If N<M is because
128 *                         the user is providing a reduced SVD as input.
129 *               on exit : The rank of A.  M >= N >= 0.
130 *
131 *  TOL         (input) real
132 *                  Tolerance to consider two singular values in the
133 *              same cluster in step 1. That is,
134 *              if    (ABS(S(I)) - ABS(S(I+1)) )/ ABS(S(I)) .LE. TOL
135 *                      then    S(I+1) belong to the same cluster as S(I).
136 *
137 *  TOLGAP      (input)  REAL.    (TOLGAP <1)
138 *              If relgap(sigma)/relgap(1) < TOLGAP
139 *              the two corresponding singular value clusters are
140 *              candidates to be in a single cluster, with the
141 *              previous or following one, in order to do the
142 *              e-vector calculation. (See step 3).
143 *
144 *  U            (input/output) REAL array, dimension (LDA, M)
145 *               On entry, the leading MxN part contains the orthonormal
146 *                         left singular vectors of the matrix A.
147 *               On exit, contains the M orthonormal eigenvectors
148 *                          of the matrix A.
149
150 *  S            (input/output) REAL, dimension (M)
151 *               On entry,  the singular values.
152 *                          If ORDER='Y', in descending order.
153 *               All singular values smaller than SLAMCH('U') are made zero.
154 *               On exit,   the M eigenvalues in descending order.
155 *               If M>N M-N zero eigenvalues are added
156 *               with their corresponding eigenvectors (if JOBVECTOR='Y')
157 *
158 *  V            (input/output) REAL array, dimension (LDA, M)
159 *               On entry,  the leading MxN part  contains the
160 *                          orthonormal right singular vectors
```

```
*                          of the matrix A.
*                Unchanged on exit if order = 'Y'.
*
*
*  INCLUST       (output/workspace) INTEGER, dimension (LDA, 6)
*                The information of the sing. value clusters
*                (clusters can have  one or more sing. val).
*                Supposed the sing. val with the usual ordering
*                sigma_i >= sigma_(i+1).
*                INCLUST(I,1) = Position of the first sing. val
*                               of the i-th cluster.
*                INCLUST(1,1) = Number of clusters
*                               (because the first cluster always begins
*                               at sigma_1)
*                               (If ISCLUST(1,1)=N there are no
*                               multidimensional clusters).
*                INCLUST(I,2) = If there are pos. evalues in cluster I,
*                               INCLUST(I,2) is the ordinal of the
*                               corresponding evalue cluster.
*                               (with the evalues ordered decreasingly,
*                               so the first cluster or evalues has the
*                               largest, no absolute values, evalues).
*                               If there are NO positive evalues in
*                               cluster I,  INCLUST(I,2)=0
*                INCLUST(I,3)   The same as INCLUST(I,2) for negative
*                               evalues in cluster I.
*                INCLUST(I,4)   Number of negative evalues in the i-th cluster
*                INCLUST(I,5)   Number of positive evalues in the i-th cluster
*                INCLUST(I,6)   Workspace
*                INCLUST(I,7)   Workspace
*                INCLUST(I,8)   Workspace
*
*  ILCLUST       (output) INTEGER, dimension N.
*                Same as INCLUST(I,1), but for clusters of evalues.
*                ILCLUST(I) = Position of the first evalue of the
*                             i-th cluster of evalues.
*                ILCLUST(1) = Number of evalue clusters
*                             (because the first cluster always begins
*                              at lambda_1)
*
*  MAXDIMCLUST   (output) INTEGER
*                The size of the biggest (singular value) cluster.
*
*  RGAPS         (output) REAL array, dimension (N,3)
*                The  relative gaps of the clusters of singular
*                values, all the s-values in a cluster share this
*                number. Ordered as the sing. values. RGAPS(I)
*                is the relgap of cluster I.
*
*  RGAPSMIN      (output) REAL
*                The  minimum relative gap of the clusters of
*                singular values.
*
*  WRK1          (workspace) REAL array, dimension  5*LDA
```

```
215 *
216 *  WRK2          (workspace) REAL array, dimension (LDA, N+1)
217 *
218 *  WRK3          (workspace) REAL array, dimension (LDA, N+1)
219 *
220 *  IKEY          (workspace) INTEGER array dimension LDA
221 *
222 *  INFO     (output) INTEGER
223 *           = 0:  successful exit
224 *
225 *  ============================================================
226 *
227 * External links
228 c     BLAS, LAPACK, SCALAPACK
229 *
230
231 *     .. Parameters ..
232       REAL                ZERO, ONE , TWO
233       PARAMETER           ( ZERO = 0.0E0, ONE = 1.0E0 , TWO = 2.0E0)
234 *     .. Local Scalars ..
235       LOGICAL             NOORDER, WANTREFINE, EVECTORS, REDUCED
236       INTEGER             I0 ,IN ,I , J , K, L, NEG, NEG1, ISC ,ILC , NI
237       INTEGER             I2 , I3 , J2 , J3 , IX , IN2
238       INTEGER             LM, LP, LP0, LPF, IFLAGM, IFLAGP, ILCPOS
239       INTEGER             IC0, ICF, IMAX
240       REAL                TRACE, SCP, SDOT
241       REAL                AUX1, AUX2, MINRGAPS, TOLGAP2
242
243 *     .. External Functions ..
244       LOGICAL             LSAME,SIGNEDCLOSEC
245       EXTERNAL            LSAME
246 *
247 *    External Subroutines ..
248 *     .....BLAS
249       EXTERNAL              SSWAP, SGEMM, SCOPY
250 *     .....LAPACK
251       EXTERNAL              SSYEV, XERBLA, SLACPY, SGEQRF, SORGQR, SGEQPF
252 *     .....SCALAPACK
253       EXTERNAL              SLASRT2
254 *     .....AUXILIARY
255       EXTERNAL          SLASWPC, PERSLASWP
256 *     ..
257 *     .. Intrinsic Functions ..
258       INTRINSIC           NINT, REAL,  MIN,  MAX, ABS
259
260 *     ..
261 *     .. Executable Statements ..
262 *
263 *     Test the input parameters.
264 *
265       EVECTORS = LSAME( JOBVECTOR, 'Y' )
266       WANTREFINE= LSAME( REFINE, 'Y' )
267 *      REREFINE= 0
268         IF (.NOT.EVECTORS) WANTREFINE=0
```

```
269         NOORDER = LSAME( ORDER, 'N' )
270         INFO = 0
271         IF( .NOT.( EVECTORS .OR. LSAME( JOBVECTOR, 'N' ) ) ) THEN
272         INFO = -1
273         ELSEIF( .NOT.( WANTREFINE .OR. LSAME( REFINE, 'N' ) ) ) THEN
274         INFO = -2
275         ELSEIF( .NOT.( NOORDER .OR. LSAME( ORDER, 'Y' ) ) ) THEN
276         INFO = -3
277         ELSEIF( LDA.LT.MAX( 1, M ) ) THEN
278         INFO = -4
279         ELSEIF( M.LT.0 ) THEN
280         INFO = -5
281         ELSEIF( (N.GT.M).OR.(N.LE.0) ) THEN
282         INFO = -6
283         ELSEIF( TOL.GE.ONE ) THEN
284         INFO = -7
285         ELSEIF( TOLGAP.GE.ONE ) THEN
286         INFO = -8
287         END IF
288         IF( INFO.NE.0 ) THEN
289            CALL XERBLA( 'SSYSVD␣', -INFO )
290            RETURN
291         END IF
292 *
293 *     Check if the input svd is reduced
294
295            REDUCED=0
296            IF(N.LT.M) THEN
297            REDUCED=1
298                    DO 11 I=N+1,M
299              S(I)=ZERO
300 11         CONTINUE
301            END IF
302 *
303 *   ********************************************************************
304 *     Steps 1 and 2: (BEGIN)
305 *         1. Decide the singular value clusters (First selection). Tolerance TOL.
306 *         2. Compute the eigenvalues.
307 *   vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
308 *
309 *      If necessary, order the singular values and vectors
310 *
311            IF (NOORDER) THEN
312              DO 12 I= 1,N
313                    IKEY(I)=I
314 12           CONTINUE
315              CALL SLASRT2 ('D',N,S,IKEY,INFO)
316             CALL SLACPY ('A',M,N,U,LDA,WRK2,LDA)
317             DO 13 I= 1,N
318                         CALL SSWAP(M,U(1,I),1,WRK2(1,IKEY(I)),1)
319 13           CONTINUE
320              CALL SLACPY ('A',M,N,V,LDA,WRK2,LDA)
321             DO 14 I= 1,N
322                    CALL SSWAP(M,V(1,I),1,WRK2(1,IKEY(I)),1)
```

```
323  14          CONTINUE
324         END IF
325  *
326  *      Assign zero value to the singular values underflow
327  *      and check if all the singular values are zero.
328  *      From now on N is the rank of A
329  *
330         DO 15 WHILE (S(N).LT.SLAMCH('U'))
331               S(N)=ZERO
332               N=N-1
333  *
334  *      If all the sing. values are zero there is only one cluster
335  *
336                     IF (N.EQ.ZERO) THEN
337                                    ISC=1
338                                    ILC=1
339                     ILCLUST(1)=1
340                       INCLUST(1,2)=0
341                       INCLUST(1,3)=0
342                       GOTO 1000
343                  ENDIF
344               REDUCED=1
345  15     CONTINUE
346  *
347  *      Initialize counters
348  *
349         LP=0
350         LPF=0
351         LM=N+1
352         LMF=LM
353         IFLAGP=0
354         IFLAGM=0
355         MAXDIMCLUST = 0
356         ILC=0
357         ISC=0
358         I0 = 0
359         TRACE = ZERO
360         NEG1 = 0
361         DO 16 I = 1 , N
362              INCLUST(I,1) = 0
363              INCLUST(I,2) = 0
364              INCLUST(I,3) = 0
365   16    CONTINUE
366         INCLUST(1,1)=1
367         ILCLUST(1)=1
368  *
369  *      Main Eigenvalue Loop
370  *      First cluster selection
371  *
372         DO 20 I = 1,N
373              SCP = SDOT(M, V(1,I) , 1 , U(1,I) , 1)
374  *
375  *      The sign of the scalar product <v(:,I), u(:,I)> is assigned
376  *      by default to the sing. value S(I). When S(I) belongs to a
```

```
377  *     multiple cluster this sign will have to be confirmed by the
378  *     count of negative and positive evalues (counter NEG1)
379  *
380           S(I) = SIGN(S(I),SCP)
381           IF (SCP.LT.ZERO) THEN
382  *
383  *             Number of negative evalues in the cluster
384  *
385                NEG1 = NEG1 + 1
386  *
387  *             Index of last negative evalue in the cluster
388  *
389                LM=LM-1
390  *
391  *             There is at least one neg. evalue in the cluster
392  *
393                IFLAGM=1
394             ELSE
395  *
396  *            Index of last positive evalue in the cluster
397  *
398           LP=LP+1
399                IF(IFLAGP.EQ.0) THEN
400  *
401  *                   Index of first positive evalue in the cluster
402  *
403                      LP0=LP
404  *
405  *                   There is at least one neg. evalue in the cluster
406  *
407                      IFLAGP=1
408                ENDIF
409           ENDIF
410  *
411  *     Trace
412  *
413          TRACE = TRACE + SCP
414  *
415  *     Condition for ending a cluster
416  *
417          IF ( (I .EQ. N) .OR.
418         $    ( (ABS(S(I)) - ABS(S(I+1)) )/ ABS(S(I)) .GT. TOL) ) THEN
419  *
420  *     Starting the counters
421  *
422  *          Dimension of the actual cluster
423  *
424           NI=I-I0
425  *
426  *          Dimension of the biggest  cluster
427  *
428           MAXDIMCLUST=MAX( MAXDIMCLUST,NI)
429  *
430  *          Number of  clusters of sing-values
```

```
*
            ISC = ISC + 1
            IF (I .NE. N) INCLUST( ISC + 1,1 ) = I+1
*           If there is at least one positive/negative evalue in the
*           cluster their indexes are
*
            IF (IFLAGP.EQ.1) INCLUST(ISC,2)=LP0
            IF (IFLAGM.EQ.1) INCLUST(ISC,3)=LM
*
*           If there is more than one sing-value in the cluster
*           the signs have to be determined
*
            IF ( (NI).NE.1) THEN
*
*       Double checking the value of the trace
*
              NEG = NINT( (REAL(I-I0) - TRACE)/TWO)
              NEG = MIN( MAX( NEG , 0), I-I0 )
              IF (NEG.NE.NEG1) THEN
                NEG1=NEG
                DO 21 J = (I0+1), (I0+NEG)
                  S(J) = -ABS(S(J))
 21             CONTINUE
                DO 22 J=I0+NEG+1,I
                  S(J) = ABS(S(J))
 22             CONTINUE
                      IF((NI-NEG).EQ.0)THEN
                  INCLUST(ISC,2)=0
                  LP=LPF
                ELSE
                  INCLUST(ISC,2)=LPF+1
                  LP=LPF+NI-NEG
                ENDIF
                      IF(NEG.EQ.0)THEN
                  LM=LMF
                  INCLUST(ISC,3)=0
                ELSE
                  LM=LMF-NEG
                  INCLUST(ISC,3)=LM
                ENDIF
              ENDIF
            ENDIF
            LPF=LP
            LMF=LM
            INCLUST( ISC, 4)= NEG1
            INCLUST( ISC, 5)= NI-NEG1
            IFLAGP=0
            IFLAGM=0
            TRACE = ZERO
            I0 = I
            NEG1 = 0
          ENDIF
 20     CONTINUE
*
```

```
485 *     Gathering the information of the clusters of evalues from the
486 *     clusters of sing. values
487 *
488       ILC=0
489 *
490 *     For positive
491 *
492       DO 31 I=1,ISC
493        IF (INCLUST(I,2).NE.0) THEN
494 *
495 *        Number of clusters of evalues
496 *
497          ILC=ILC+1
498 *
499 *        Evalue index of the first (positive) evalue in  the cluster
500 *
501          ILCLUST(ILC)=INCLUST(I,2)
502 *
503 *        Positive evalue cluster number ILC is in sing. value cluster number I
504 *
505          INCLUST(I,2)=ILC
506        ENDIF
507 31    CONTINUE
508       IF(M-N.GT.0) ILCPOS=ILC
509 *
510 *     And for negative
511 *
512       DO 32 I=ISC,1,-1
513          IF (INCLUST(I,3).NE.0) THEN
514            ILC=ILC+1
515            ILCLUST(ILC)=INCLUST(I,3)
516            INCLUST(I,3)=ILC
517          ENDIF
518 32    CONTINUE
519 *
520 *  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
521 *    Steps 1 and 2: (END)
522 *        1. Decide the singular value clusters (First selection). Tolerance TOL.
523 *        2. Compute the eigenvalues.
524 *  ***************************************************************
525
526 *
527 *     The evalues are ordered by modulus |S(I)| GEQ |S(I+1)|,
528 *     Order them by magnitude, and store them in the temporary
529 *     array WRK2(I,1) GEQ WRK2(I+1,1)
530 *
531       DO 33 I=1,N
532            WRK2(I,1)=S(I)
533 33    CONTINUE
534          INFO=0
535       DO 34 I= 1,N
536            IKEY(I)=I
537 34       CONTINUE
538          CALL SLASRT2( 'D', N, WRK2(1,1), IKEY, INFO )
```

```
539 *
540 *     Singular value relgaps for each cluster.
541 *     They are indexed as the sing. value clusters
542 *     RGAPS(I,1) >0 if it is got from clusters (I,I+1)
543 *     RGAPS(I,1) <0 if it is got from clusters  (I-1,I)
544 *
545       RGAPSMIN = TWO
546       IF(ISC.EQ.1) THEN
547           RGAPS(1,1)=1.0E0
548       ELSE
549         DO 35 I=1,ISC
550           IF(I.EQ.1) THEN
551             IN =  INCLUST(2,1)
552             RGAPS(I,1) = MIN(1.,
553      %                    ABS ( (ABS(S(IN-1))-ABS(S(IN)))/S(IN-1) ))
554           ELSEIF(I.EQ.ISC) THEN
555             IN =  INCLUST(ISC,1)
556             RGAPS(I,1) = -MIN(1.,
557      %                     ABS ( (ABS(S(IN-1))-ABS(S(IN)))/S(IN) ))
558           ELSE
559             IN =  INCLUST(I+1,1)
560             RGAPS(I,1) = MIN(1.,
561      %                    ABS ( (ABS(S(IN-1))-ABS(S(IN)))/S(IN-1) ))
562             IN =  INCLUST(I,1)
563             AUX = MIN(1.,
564      %              ABS ( (ABS(S(IN-1))-ABS(S(IN)))/S(IN) ) )
565             IF(AUX.LT.RGAPS(I,1))   RGAPS(I,1) = - AUX
566           ENDIF
567           RGAPSMIN= MIN (RGAPSMIN,ABS(RGAPS(I,1)))
568 35      CONTINUE
569       ENDIF
570
571 *  ********************************************************************
572 *    Step 3: (BEGIN)
573 *        3. Merge, when necessary, some pairs of clusters (Second selection).
574 *           Tolerance TOLGAP.
575 *  vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
576
577 *     Given the clusters obtained from the evalue calculation modify
578 *     INCLUST to take into account which pairs of clusters (one whole
579 *     positive, the other negative) can be joined to improve the
580 *     calculation of  evectors.
581 *     See [1], Sec. 5
582 *
583       IF(ISC.EQ.1) THEN
584           WANTREFINE=0
585       ENDIF
586
587 *     Counter for the second refining
588       IN2=0
589
590       IF(WANTREFINE) THEN
591 *
592 *     Get the quotient relgap(evalue)/relgap(sing.value), but only:
```

```
593 *     - For whole positive or whole negative (signed) sing value cluster.
594 *     - If the clusters I0 and closer cluster(I0+1 or I0-1) are
595 *       signed differently
596 *     If the sing. value cluster is not signed this quotient is made TWO.
597 *     They are indexed as the sing. value clusters.
598 *
599       MINRGAPS = TWO
600       DO 36 I=1,ISC
601          I2=INCLUST(I,2)
602          I3=INCLUST(I,3)
603          IF (I.EQ.1) THEN
604             J2=INCLUST(2,2)+I3
605             J3=INCLUST(2,3)+I2
606          ELSEIF (I.EQ.ISC) THEN
607             J2=INCLUST(ISC-1,2)+I3
608             J3=INCLUST(ISC-1,3)+I2
609          ELSE
610             J2=INCLUST(I+SIGN(ONE, RGAPS(I,1)),2)+I3
611             J3=INCLUST(I+SIGN(ONE, RGAPS(I,1)),3)+I2
612          ENDIF
613          IF ((I2*I3.EQ.0).and.(j2*j3.eq.0)) THEN
614                IF (J2.EQ.0)THEN
615                   IX=I2
616                ELSEIF (J3.EQ.0) THEN
617                   IX=I3
618                ENDIF
619              IL0=ILCLUST(IX)
620 *
621 *     Initially store relgap(evalue) in WRK1S
622 *
623              IF(IX.EQ.1) THEN
624                   ILF=ILCLUST(IX+1)-1
625                   WRK1(I) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))/WRK2(ILF,1))
626              ELSEIF(IX.EQ.ILC) THEN
627                   WRK1(I) = ABS((WRK2(IL0-1,1)-WRK2(IL0,1))/WRK2(IL0,1))
628              ELSE
629                   ILF=ILCLUST(IX+1)-1
630                   WRK1(I) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))/WRK2(ILF,1))
631                   WRK1(I)= MIN(WRK1(I),
632       %           ABS((WRK2(IL0-1,1)-WRK2(IL0,1))/WRK2(IL0,1)))
633              ENDIF
634 *
635 *     Finally store relgap(sing.value)/relgap(evalue) in WRK1
636 *
637              WRK1(I)=MIN(1.,WRK1(I))
638              WRK1(I)=ABS(RGAPS(I,1)/WRK1(I))
639          ELSE
640            WRK1(I)=TWO
641          ENDIF
642          IF (WRK1(I).LT.MINRGAPS) THEN
643             MINRGAPS=WRK1(I)
644             I0=I
645          ENDIF
646 36     CONTINUE
```

```
647
648           DO 40 WHILE (MINRGAPS.LT.TOLGAP)
649  *
650  *           Do not consider again this cluster as a candidate to join
651  *
652               WRK1(I0)    =  TWO
653  *
654  *            Determine the candidate for joining:  either I0+1 or I0-1
655  *
656               IF(RGAPS(I0,1).LT.ZERO) THEN
657                     I0=I0-1
658               SIGNEDCLOSEC=(INCLUST(i0,2)*INCLUST(i0,3).eq.0)
659               else
660                SIGNEDCLOSEC=(INCLUST(i0+1,2)*INCLUST(i0+1,3).eq.0)
661               ENDIF
662  *
663  *           Get the sing. value relgap of the new cluster (I0,I0+1)
664  *
665               IC0=INCLUST(I0,1)
666               IF(I0.EQ.1) THEN
667                    ICF=INCLUST(3,1)-1
668                    AUX1 = MIN(1.,
669           %                 ABS ( (ABS(S(ICF))-ABS(S(ICF+1)))/S(ICF) ))
670               ELSEIF(I0.EQ.(ISC-1)) THEN
671                    AUX1 = -MIN(1.,
672           %                  ABS ( (ABS(S(IC0-1))-ABS(S(IC0)))/S(IC0)))
673               ELSE
674                    ICF=INCLUST(I0+2,1)-1
675                    AUX1= MIN(1.,
676           %                ABS ( (ABS(S(ICF))-ABS(S(ICF+1)))/S(ICF) ))
677                    AUX2= MIN(1.,
678           %                ABS ( (ABS(S(IC0-1))-ABS(S(IC0))) /S(IC0)))
679                    IF(AUX2.LT.AUX1) AUX1 = -AUX2
680               ENDIF
681  *
682  *           Compare it with the two previous ones
683  *
684               AUX2 = MIN( ABS(RGAPS(I0,1)),ABS(RGAPS(I0+1,1)))
685  *
686  *           If it is better, we join  clusters I0 and I0+1 in a new one
687  *
688                if (SIGNEDCLOSEC) then
689               IF((ABS(AUX1)/AUX2).GT.ONE) THEN
690                    INCLUST(I0,2)=MAX(INCLUST(I0,2),INCLUST(I0+1,2))
691                    INCLUST(I0,3)=MAX(INCLUST(I0,3),INCLUST(I0+1,3))
692                    INCLUST(I0,4) = INCLUST(I0,4)+INCLUST(I0+1,4)
693                    INCLUST(I0,5) = INCLUST(I0,5)+INCLUST(I0+1,5)
694                    RGAPS(I0,1) = AUX1
695                    ISC = ISC-1
696                   DO 41 J=I0+1, ISC
697                        INCLUST(J,1) = INCLUST(J+1,1)
698                        INCLUST(J,2) = INCLUST(J+1,2)
699                        INCLUST(J,3) = INCLUST(J+1,3)
700                        INCLUST(J,4) = INCLUST(J+1,4)
```

```
                    INCLUST(J,5) = INCLUST(J+1,5)
                    RGAPS(J,1)   = RGAPS(J+1,1)
                    WRK1(J)      = WRK1(J+1)
 41             CONTINUE
*
*           Do not consider again this cluster as a candidate to join
*
                WRK1(I0)   =  TWO
                  IF(I0.EQ.ISC) THEN
                    IMAX= N -INCLUST(ISC,1)
                ELSE
                    IMAX= INCLUST(I0+1,1) -INCLUST(I0,1)
                ENDIF
                  IF(IMAX.GT.MAXDIMCLUST) MAXDIMCLUST=IMAX
              ENDIF
              ENDIF
            MINRGAPS=TWO
            DO 42 J = 1, ISC
               IF (WRK1(J).LT.MINRGAPS) THEN
                   MINRGAPS=WRK1(J)
                   I0=J
               ENDIF
 42         CONTINUE
*
*         Go for the next candidate to join
*
 40      CONTINUE
*
*        Get the minimum relgap of all the clusters of sing. values
*
         RGAPSMIN=ABS(RGAPS(1,1))
         DO 51 J = 1, ISC
               IF (ABS(RGAPS(J,1)).LT.RGAPSMIN) THEN
                   I0=J
                   RGAPSMIN=ABS(RGAPS(J,1))
               ENDIF
 51      CONTINUE
         ILCLUST(1)=ILC




*    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*      Step 3: (END)
*          3. Merge, when necessary, some pairs of clusters (Second selection).
*             Tolerance TOLGAP.
*    *******************************************************************


*    *******************************************************************
*      Step 5: (BEGIN)
*          5. Make a new selection of clusters (Third selection)
*          (Without eigenvectors yet)
*    vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

```
755
756 *    Eigvalues relgaps for each cluster.
757 *
758         TOLGAP2=1/(TWO*(N-1))
759         DO 60 I=1,ISC
760              I2=INCLUST(I,2)
761              I3=INCLUST(I,3)
762 *
763 *              Positive eigvalues stored in RAGAPS(I,2)
764 *
765                   IF (I2.GT.0)THEN
766                      IL0=ILCLUST(I2)
767                     IF(I2.EQ.1) THEN
768                     ILF=ILCLUST(I2+1)-1
769                     RGAPS(I,2) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))
770         %                    /WRK2(ILF,1))
771                   ELSEIF(I2.EQ.ILC) THEN
772                     RGAPS(I,2) = ABS((WRK2(IL0-1,1)-WRK2(IL0,1))
773         %                    /WRK2(IL0,1))
774                   ELSE
775                     ILF=ILCLUST(I2+1)-1
776                     RGAPS(I,2) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))
777         %                    /WRK2(ILF,1))
778                     RGAPS(I,2)= MIN(RGAPS(I,2),
779         %                ABS((WRK2(IL0-1,1)-WRK2(IL0,1))/WRK2(IL0,1)))
780                   END IF
781                    RGAPS(I,2)= MIN(RGAPS(I,2),ONE)
782                 ELSE
783                     RGAPS(I,2)=ZERO
784                 END IF
785 *
786 *              Negative eigvalues intially stored in WRK1
787 *
788                   IF (I3.GT.0) THEN
789                      IL0=ILCLUST(I3)
790                      IF(I3.EQ.1) THEN
791                     ILF=ILCLUST(I3+1)-1
792                     WRK1(I) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))
793         %                    /WRK2(ILF,1))
794                    ELSEIF(I3.EQ.ILC) THEN
795                     WRK1(I) = ABS((WRK2(IL0-1,1)-WRK2(IL0,1))
796         %                    /WRK2(IL0,1))
797                    ELSE
798                     ILF=ILCLUST(I3+1)-1
799                     WRK1(I) = ABS((WRK2(ILF,1)-WRK2(ILF+1,1))
800         %                    /WRK2(ILF,1))
801                     WRK1(I)= MIN(WRK1(I),
802         %                ABS((WRK2(IL0-1,1)-WRK2(IL0,1))/WRK2(IL0,1)))
803                     ENDIF
804                     WRK1(I)= MIN(WRK1(I),ONE)
805                   ELSE
806                       WRK1(I)=ZERO
807                   ENDIF
808
```

```
809 *               Store the minimun rgap(positive or negative)in RGAPS(I,2)
810 *              signing it minus if comes from the negative eigvalue
811
812                  IF (RGAPS(I,2).LT.WRK1(I)) THEN
813                    RGAPS(I,2)=-WRK1(I)
814                  END IF
815 *
816 *             Anotate if there is a cluster that has triggered the tolerance
817                IF (ABS(RGAPS(I,1)/RGAPS(I,2)).LT.TOLGAP2) THEN
818
819 *              We only consider clusters whose closest clusters are signed one way
820                 J=I+ SIGN(ONE,RGAPS(I,1))
821                 IF (INCLUST(J,4)* INCLUST(J,5).EQ.0) THEN
822                  IN2=IN2+1
823                  INCLUST2(IN2,1)=I
824                 END IF
825                END IF
826
827 60        CONTINUE
828 *
829 *       If some cluster has triggered the tolerance...
830 *
831           IF (IN2.GT.0) THEN
832             DO 70 I=1,IN2
833                I0=INCLUST2(I,1)
834                INCLUST2(I,2)=0
835                INCLUST2(I,3)=0
836
837                AUX2=ABS(RGAPS(I0,2)*TOLGAP2)
838
839 *
840 *            Look on the rigth to find r' (right boundary)
841 *
842                IF (I0.GT.1) THEN
843                    J=0
844                    AUX1=(ABS(S(INCLUST(I0-J,1)-1))-ABS(S(INCLUST(I0-J,1))))
845       %               /ABS(S(INCLUST(I0-J,1)))
846                   DO 71 WHILE (AUX1.LT.AUX2)
847                    J=J+1
848                    IF (I0-J.GT.1) THEN
849                    AUX1=(ABS(S(INCLUST(I0-J,1)-1))-ABS(S(INCLUST(I0-J,1))))
850       %               /ABS(S(INCLUST(I0-J,1)))
851                   ELSE
852                    AUX1=AUX2
853                   END IF
854
855 71             CONTINUE
856                 INCLUST2(I,3)=J
857 *                IF (J.GT.0) REREFINE=1
858                END IF
859 *
860 *            Look on the left to find l' (left boundary)
861 *
862                IF (I0.LT.ISC) THEN
```

```
                 J=0
                 AUX1=(ABS(S(INCLUST(I0+J+1,1)-1))-ABS(S(INCLUST(I0+J+1,1))))
      %                /ABS(S(INCLUST(I0+J+1,1)-1))
                 DO 72 WHILE (AUX1.LT.AUX2)
                 J=J+1
                 IF (I0+J.LT.ISC) THEN
                 AUX1=(ABS(S(INCLUST(I0+J+1,1)-1))-ABS(S(INCLUST(I0+J+1,1))))
      %                /ABS(S(INCLUST(I0+J+1,1)-1))
                 ELSE
                 AUX1=AUX2
                 END IF
 72              CONTINUE
                 INCLUST2(I,2)=J
*                  IF (J.GT.0) REREFINE=1
               END IF
 70          CONTINUE
           END IF

*          IF(WANTREFINE) THEN
         END IF

*  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*     Step 5: (BEGIN)
*         5. Make a new selection of clusters (Third selection)
*         (Without eigenvectors yet)
*  ************************************************************************


*  ************************************************************************
*     Step 4: (BEGIN)
*         4. Compute the eigenvectors
*  vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

         IF(EVECTORS) THEN
           CALL SLACPY ('A',M,N,U,LDA,WRK3,LDA)
           CALL SLACPY ('A',M,1,S,LDA,WRK3(1,M+1),LDA)
                DO 80 I=1,ISC
                     I0=INCLUST(I,1)
                          IF(I.EQ.ISC) THEN
                               NI=N+1
                          ELSE
                               NI=INCLUST(I+1,1)
                          ENDIF
                          NI=NI-I0
                          IF (INCLUST(I,2)*INCLUST(I,3).NE.0) THEN

                            CALL SGEMM('T','N',NI,NI,M,ONE,V(1,I0),LDA,
      $                              U(1,I0),LDA,ZERO,WRK2,LDA)
*
*        Diagonalize  V(1:M,I0+1:I)^T U(1:M,I0+1:I) using LAPACK SSYEV
*
                            CALL SSYEV('V','U',NI,WRK2,LDA,WRK2(I0,M+1)
      $                        ,WRK1,3*NI,INFO)

```

```
*
*      Store the eigenvectors in U
*
                        CALL SGEMM('N', 'N', M, NI, NI, ONE, V(1,I0),
     $                  LDA, WRK2, LDA, ZERO, U(1,I0), LDA)

c      Order the eigvalues to agree with the eigvectors ("increasingly")

                      DO 81 J= 1,NI
                                    IKEY(J)=J
81                      CONTINUE

                        CALL SLASRT2('I',NI,S(I0),IKEY,INFO)

*                   If all the evalues in the cluster have the same sign
*                   there is nothing left to do
*
                      ELSE
                          DO 82 J=I0,I0+NI-1
                            CALL SCOPY(M, V(1,J),1, U(1,J),1)
82                        CONTINUE
                      ENDIF
80              CONTINUE

*    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*     Step 4: (END)
*         4. Compute the eigenvectors
*    **********************************************************************


*    **********************************************************************
*     Step 5: (BEGIN)
*         5. With the new clusters of the third cluster selection refine
*            some eigenvectors
*    vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

*       IF (REREFINE) THEN
        IF (WANTREFINE.AND.(IN2.GT.0)) THEN
          DO 90 I=1,IN2

c       STEP 1: Compute the  Z+ y Z- vectors from the macrocluster
              IN=INCLUST2(I,1)
              IL= IN + INCLUST2(I,2)
              IR= IN - INCLUST2(I,3)

c             Index I0: where the macrocluster begins
              IF (IR.EQ.1) THEN
                 I0=1
              ELSE
                 I0=INCLUST(IR,1)
              END IF
c             Index IE: where the macrocluster ends
              IF(IL.EQ.ISC) THEN
                 IE=N+1
```

```
971                ELSE
972                  IE=INCLUST(IL+1,1)
973                ENDIF
974   c            NI: macrocluster´s length
975                NI=IE-I0
976
977   c            Compute U^TV correposponding to the macroluster
978                CALL SGEMM('T','N',NI,NI,M,ONE,V(1,I0),LDA,
979         $                       WRK3(1,I0),LDA,ZERO,WRK2,LDA)
980
981   c            Diagonalize U^TV
982                CALL SSYEV('V','U',NI,WRK2,LDA,WRK2(1,M+1),WRK1,
983         $                          3*NI,INFO)
984
985   c            Z vectors stored in WRK3
986                CALL SGEMM('N', 'N', M, NI, NI, ONE, V(1,I0),
987         $                      LDA, WRK2, LDA, ZERO, WRK3(1,I0), LDA)
988
989   c            Sort the Z vectors increasingly according
990   C                                  to the singular values (:,n+1)
991                DO 91 J= 1,NI
992                   IKEY(J)=J
993   91           CONTINUE
994                CALL SLASRT2('I',NI,WRK3(I0,M+1),IKEY,INFO)
995
996   c            Sort the Q eigenvectors increasingly according
997   C                                       to the singular values
998                DO 92 J= 1,NI
999                  IKEY(J)=J
1000  92           CONTINUE
1001
1002               CALL SLASRT2('I',NI,S(I0),IKEY,INFO)
1003               CALL PERSLASWP(NI, LDA, IKEY, INCLUST(1,6),
1004        $                INCLUST(1,7), INCLUST(1,8))
1005               CALL SLASWPC(N,U(1,I0),LDA,1,NI-1,INCLUST(1,6),1)
1006
1007  c        STEP 2: Compute the P matrix as the Q factor (QR decomposition)
1008  c            of  Z-´*(Q-*permQ) (resp)
1009
1010  c            Adjust the index depending on whether the core
1011  C                                          is positive or negative
1012               IF (RGAPS(IN,2).LT.ZERO) THEN
1013                 NCORE = INCLUST(IN,4)
1014                 NPATH = NCORE
1015                 NFLAG = 0
1016               ELSE
1017                 NCORE = INCLUST(IN,5)
1018                 NPATH = 0
1019                 NFLAG = 1
1020               END IF
1021
1022  c            Sort Zt*Q- "out␣of␣the␣core" vectors in WRK2
1023               CALL SGEMM('T','N',NI-NCORE,NI-NCORE,M,ONE,WRK3(1,
1024        $                I0+NPATH),LDA,U(1,I0+NPATH),LDA,ZERO,WRK2,LDA)
```

```
1025  c              Permute the matrix Z^T*Q- according to decreasing
1026  c                                                      relgaps in WRK2
1027  c                      First get the rigth order of the relgaps
1028                  L=0
1029                  DO 93 J=IL,IR,-1
1030                     IF (J.EQ.IN) THEN
1031                        DO 931 K=1,(INCLUST(J,4)+INCLUST(J,5)-NCORE)
1032                           L=L+1
1033                           RGAPS(L,3)=ABS(RGAPS(J,1))
1034  931                   CONTINUE
1035                     ELSE
1036                        DO 932 K=1,(INCLUST(J,4)+INCLUST(J,5))
1037                           L=L+1
1038                           RGAPS(L,3)=ABS(RGAPS(J,1))
1039  932                   CONTINUE
1040                     END IF
1041  93              CONTINUE
1042
1043                  DO 439 J= 1,NI
1044                     IKEY(J)=J
1045  439             CONTINUE
1046
1047                  CALL SLASRT2('D',NI-NCORE,RGAPS(1,3),IKEY,INFO)
1048                  CALL PERSLASWP(NI-NCORE, LDA, IKEY, INCLUST(1,6),
1049        $                INCLUST(1,7), INCLUST(1,8))
1050
1051  c               Permute  Zt*Q-  (out)
1052                  CALL SLASWPC(NI-NCORE,WRK2,LDA,1,NI-NCORE-1,
1053        %                INCLUST(1,6),1)
1054
1055                  DO 94 J=I,NI
1056                   INCLUST(J,7)=ZERO
1057  94              CONTINUE
1058
1059  c               Compute A QR DECOKMPOSITION TO GET P
1060                  CALL  SGEQRF(NI-NCORE, NI-NCORE,WRK2,LDA,
1061        %                WRK2(1,NI), WRK1,-1,INFO)
1062                  CALL SORGQR(NI-NCORE,NI-NCORE,NI-NCORE,WRK2,
1063        %                        LDA, WRK2(1,NI),WRK1,-1,INFO)
1064
1065  c               Unpermute  Zt*Q-  (out)
1066                  CALL SLASWPC(NI-NCORE,WRK2,LDA,1,NI-NCORE-1,
1067        %                INCLUST(1,6),-1)
1068
1069  c          STEP 3: Get the Q Zt*Q- (out) as the negative
1070  c                                          (resp positive) eigvectors
1071                  CALL SGEMM('N','N',M, NI-NCORE,NI-NCORE,ONE,WRK3(1,
1072        $                I0+NPATH),LDA,WRK2(1,1),LDA,ZERO,U(1,I0+NPATH),LDA)
1073
1074  c          STEP 4: Get the Q Zt*Q+  (in core) as the positive
1075  c                                           (or negative) eigvectors
1076                  CALL SLACPY('A',M,NCORE,WRK3(1,I0+(NI-NCORE)*NFLAG)
1077        $                ,LDA,U(1,I0+(NI-NCORE)*NFLAG),LDA)
1078                  CALL SLASRT2('I',NI,wrk3(I0,m+1),IKEY,INFO)
```

```
                 CALL SCOPY(NI, WRK3(I0,m+1),1, S(I0),1)
 90           CONTINUE

*            IF WANTREFINE.AND.(IN2.GT.0) THEN
*          IF (REREFINE) THEN
          END IF


*   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*     Step 5: (END)
*          5. With the new clusters of the third cluster selection refine
*             some eigenvectors
*   **********************************************************************



*   **********************************************************************
*    If there are zero eigenvalues, compute the corresponding eigenvectors
*    and add a cluster of zero eigenvalues (BEGIN)
*   vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

              IF(REDUCED) then
                  DO 95 I=1,N
                     CALL SCOPY(M, V(1,I),1, WRK2(1,I),1)
 95               CONTINUE
                  INFO=0
*
*             Apply QR with column pivoting to get the evectors
*             corresponding to the zero evalue
*
                  CALL SGEQPF( M, N, WRK2, LDA, IKEY, V(1,N+1), WRK1,INFO)
                  IF( INFO.NE.0 ) THEN
                     CALL XERBLA( 'SGEQPF␣IN␣SSYSVD␣', 2000+INFO )
                     RETURN
                  ENDIF
                  INFO=0
*
*              The evectors are temporarily stored in the last
*              M-N columns of WRK2
*
                  CALL SORGQR( M, M, N, WRK2, LDA, V(1,N+1), WRK1,M,INFO)
                  IF( INFO.NE.0 ) THEN
                      CALL XERBLA( 'SORGQR␣IN␣SSYSVD␣', 3000+INFO )
                      RETURN
                  ENDIF
                  ENDIF

*         IF(EVECTORS) THEN
           ENDIF
*
*       If there are eigenvalues equal to zero make the last cluster with them
*

          IF (REDUCED) THEN
```

```
            ISC=ISC+1
               INCLUST(ISC,1)=N+1
               INCLUST(ISC,2)=0
               INCLUST(ISC,3)=0
               IF(ILCPOS.NE.ILC) THEN
              DO 102  I=ILC+1,ILCPOS+2,-1
                  ILCLUST(I)=ILCLUST(I-1)+M-N
102           CONTINUE
               ELSE
               ILCLUST(ILC+1)=N+1
            ENDIF
               ILC=ILC+1
               IF(EVECTORS) THEN
                DO 103  I=N+1,M
                     CALL SCOPY(M,  WRK2(1,I),1,  U(1,I),1)
103             CONTINUE
            END IF

            ENDIF

*   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*    If there are zero eigenvalues, compute the corresponding eigenvectors
*    and add  a cluster of zero eigenvalues (END)
*   **********************************************************************


*   **********************************************************************
*       Finally order evalues decreasingly (evectors accordingly) (BEGIN)
*   vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

         INFO=0
         DO 104  I= 1,M
                 IKEY(I)=I
104         CONTINUE

         CALL SLASRT2( 'D', M, S, IKEY, INFO )
         IF(EVECTORS) THEN
          CALL PERSLASWP(M, LDA, IKEY, INCLUST(1,6),
        $         INCLUST(1,7), INCLUST(1,8))
          CALL SLASWPC(M,U,LDA,1,M-1,INCLUST(1,6),1)
         ENDIF

*   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
*       Finally order evalues decreasingly (evectors accordingly) (END)
*   **********************************************************************

*       Number of sing. value clusters
1000    ILCLUST(1)=ILC
        INCLUST(1,1)=ISC

*
*       END OF   SSYSVD.
*
        RETURN
```

```
1187        END
1188
1189
1190
1191        SUBROUTINE PERSLASWP(N, LDA, PERMU, LIST, PARTIAL, DONDE)
1192
1193  *
1194  *  -- SYSVD auxiliary routine --
1195  *     Univ. Carlos III de Madrid
1196  *     Nov 2008
1197  *
1198  *  Purpose
1199  *  =======
1200  *     Given a permutation PERMU of the indices 1:N, PERSLASWP  creates
1201  *     a vector LIST(1:N-1) of elemetary interchanges such that the
1202  *     sequence of elementary interchanges K <-> LIST(J) acted upon 1:N
1203  *     produces PERMU.
1204  *     This is an auxiliary routine to use with
1205  *
1206  *     SLASWP performs a series of row interchanges on the matrix A.
1207  *     One row interchange is initiated for each of rows K1 through K2 of A.
1208  *   or
1209  *     SLASWPC (the same for columns)
1210  *
1211  *     Example
1212  *     PERMU=[3  4  5  6  2  1]
1213  *     LIST=[3  4  5  6  6]
1214
1215
1216  *     .. Scalar Arguments ..
1217        INTEGER            LDA, N
1218  *     ..
1219  *     .. Array Arguments ..
1220        INTEGER            PERMU(*), LIST(*)
1221        INTEGER            PARTIAL(*), DONDE(*)
1222
1223  *
1224  *  Arguments
1225  *  =========
1226  *
1227  *  N        (input) INTEGER
1228  *           The number of elements of PERMU.
1229  *
1230  *  LDA      (input) INTEGER
1231  *           The leading dimension of the array A.
1232  *
1233  *  PERMU    (input) INTEGER array, dimension N
1234  *           The initial permutation to be decomposed as elementary interchanges.
1235  *
1236  *  LIST     (output) INTEGER array, dimension N
1237  *           The vector of elementary interchanges, such that
1238  *           LIST(K) = L implies the K-th step in going from 1:N to PERMU
1239  *           is to interchange elementes K and L.
1240  *
```

```
1241  *  PARTIAL (work) INTEGER array, dimension N
1242  *          Vector of the partial permutations. Starting from 1:N.
1243  *          On output PARTIAL=PERMU
1244  *
1245  *  DONDE   (work/output) INTEGER array, dimension N
1246  *          The inverse vector of PERMU. It is being built at the same time
1247  *          as  PARTIAL. At the beginning DONDE=1:N,
1248  *          on output   DONDE(K)=J <-> PERMU(J)=k
1249  *
1250  *  Further Details
1251  *  ===============
1252  **
1253  *     .. Local Scalars ..
1254        INTEGER    I, J, K, DK, IAUXP, IAUXD
1255
1256        DO 1 I=1,N
1257           DONDE(I)=I
1258           PARTIAL(I)=I
1259  1     CONTINUE
1260        DO 2 J=1, N-1
1261           K=PERMU(J)
1262           DK=DONDE(K)
1263           LIST(J)=DK
1264           IAUXP=PARTIAL(J)
1265           PARTIAL(J)=PARTIAL(DK)
1266           PARTIAL(DK)=IAUXP
1267           IAUXD=DONDE(IAUXP)
1268           DONDE(IAUXP)=DONDE(PARTIAL(J))
1269           DONDE(PARTIAL(J))=IAUXD
1270  2     CONTINUE
1271  *
1272        RETURN
1273  *
1274  *     End of PERSLASWP
1275  *
1276        END
1277
1278
1279        SUBROUTINE SLASWPC( N, A, LDA, K1, K2, IPIV, INCX )
1280  *
1281  *     Modification: May 2009
1282  *     Germán Villanueva
1283  *     Univ. Carlos III de Madrid
1284  *     SLASWPC is the same as SLASWP but doing the interchanges by columns
1285  *
1286  *  -- LAPACK auxiliary routine (version 3.1) --
1287  *     Univ. of Tennessee, Univ. of California Berkeley and NAG Ltd..
1288  *     November 2006
1289  *
1290  *     .. Scalar Arguments ..
1291        INTEGER            INCX, K1, K2, LDA, N
1292  *     ..
1293  *     .. Array Arguments ..
1294        INTEGER            IPIV( * )
```

```
1295        REAL               A( LDA, * )
1296  *     ..
1297  *
1298  *  Purpose
1299  *  =======
1300  *
1301  *  SLASWP performs a series of row interchanges on the matrix A.
1302  *  One row interchange is initiated for each of rows K1 through K2 of A.
1303  *
1304  *  Arguments
1305  *  =========
1306  *
1307  *  N       (input) INTEGER
1308  *          The number of rows of the matrix A.
1309  *
1310  *  A       (input/output) REAL array, dimension (N,LDA)
1311  *          On entry, the matrix of column dimension N to which the row
1312  *          interchanges will be applied.
1313  *          On exit, the permuted matrix.
1314  *
1315  *  LDA     (input) INTEGER
1316  *          The leading dimension of the array A.
1317  *
1318  *  K1      (input) INTEGER
1319  *          The first element of IPIV for which a column interchange will
1320  *          be done.
1321  *
1322  *  K2      (input) INTEGER
1323  *          The last element of IPIV for which a column interchange will
1324  *          be done.
1325  *
1326  *  IPIV    (input) INTEGER array, dimension (K2*abs(INCX))
1327  *          The vector of pivot indices.  Only the elements in positions
1328  *          K1 through K2 of IPIV are accessed.
1329  *          IPIV(K) = L implies columns K and L are to be interchanged.
1330  *
1331  *  INCX    (input) INTEGER
1332  *          The increment between successive values of IPIV.  If IPIV
1333  *          is negative, the pivots are applied in reverse order.
1334  *
1335  *  Further Details
1336  *  ===============
1337  *
1338  *  Modified by
1339  *   R. C. Whaley, Computer Science Dept., Univ. of Tenn., Knoxville, USA
1340  *
1341  *  =====================================================================
1342  *
1343  *     .. Local Scalars ..
1344        INTEGER            I, I1, I2, INC, IP, IX, IX0, J, K, N32
1345        REAL               TEMP
1346  *     ..
1347  *     .. Executable Statements ..
1348  *
```

```
*     Interchange column I with column IPIV(I) for each of colums K1 through K2.
*
      IF( INCX.GT.0 ) THEN
         IX0 = K1
         I1 = K1
         I2 = K2
         INC = 1
      ELSE IF( INCX.LT.0 ) THEN
         IX0 = 1 + ( 1-K2 )*INCX
         I1 = K2
         I2 = K1
         INC = -1
      ELSE
         RETURN
      END IF
*
      N32 = ( N / 32 )*32
      IF( N32.NE.0 ) THEN
         DO 30 J = 1, N32, 32
            IX = IX0
            DO 20 I = I1, I2, INC
               IP = IPIV( IX )
               IF( IP.NE.I ) THEN
                  DO 10 K = J, J + 31
                     TEMP = A( K, I )
                     A( K, I ) = A( K, IP )
                     A( K, IP) = TEMP
   10             CONTINUE
               END IF
               IX = IX + INCX
   20       CONTINUE
   30    CONTINUE
      END IF
      IF( N32.NE.N ) THEN
         N32 = N32 + 1
         IX = IX0
         DO 50 I = I1, I2, INC
            IP = IPIV( IX )
            IF( IP.NE.I ) THEN
               DO 40 K = N32, N
                  TEMP = A( K, I)
                  A( K ,I ) = A(  K ,IP)
                  A(  K , IP) = TEMP
   40          CONTINUE
            END IF
            IX = IX + INCX
   50    CONTINUE
      END IF
*
      RETURN
*
*     End of SLASWPC
*
      END
```